VTT SYMPOSIUM 195

Keywords:
software process improvement, software quality management, product quality, software engineering

International Conference on Product Focused Software Process Improvement

Oulu, Finland, June 22 - 24, 1999

Edited by

Markku Oivo
VTT Electronics

Pasi Kuvaja
University of Oulu,
Department of Information Processing Science

Organised by

VTT Electronics
University of Oulu

TECHNICAL RESEARCH CENTRE OF FINLAND
ESPOO 1999

ISBN 951–38–5270–9 (soft back ed.)
ISSN 0357–9387 (soft back ed.)

ISBN 951–38–5271–7 (URL: http://www.inf.vtt.fi/pdf/)
ISSN 1455–0873 (URL: http://www.inf.vtt.fi/pdf/)

JULKAISIJA – UTGIVARE – PUBLISHER

Valtion teknillinen tutkimuskeskus (VTT), Vuorimiehentie 5, PL 2000, 02044 VTT
puh. vaihde (09) 4561, faksi 456 4374

Statens tekniska forskningscentral (VTT), Bergsmansvägen 5, PB 2000, 02044 VTT
tel. växel (09) 4561, fax 456 4374

Technical Research Centre of Finland (VTT), Vuorimiehentie 5, P.O.Box 2000, FIN–02044 VTT, Finland
phone internat. + 358 9 4561, fax + 358 9 456 4374

VTT Elektroniikka, Sulautetut ohjelmistot, Kaitoväylä 1, PL 1100, 90571 OULU
puh. vaihde (08) 551 2111, faksi (08) 551 2320

VTT Elektronik, Inbyggd programvara, Kaitoväylä 1, PB 1100, 90571 ULEÅBORG
tel. växel (08) 551 2111, fax (08) 551 2320

VTT Electronics, Embedded Software, Kaitoväylä 1, P.O.Box 1100, FIN–90571 OULU, Finland
phone internat. + 358 8 551 2111, fax + 358 8 551 2320

Libella Painopalvelu Oy, Espoo 1999

Preface

Software process improvement (SPI) has become a practical tool for companies where software quality is of prime value. This International Conference on Product Focused Software Process Improvement (PROFES'99) concentrates on professional software process improvement motivated by product quality needs. Often, this is facilitated by software process assessment, software measurement, process modelling, technology transfer, and software quality. The PROFES'99 conference presents the latest developments, not only in software process improvement, but also in how to improve the quality of the final product in a cost-effective way.

The conference theme "*Software product and professional software process improvement*" is emphasized in the approaches to software quality management and improvement. This theme addresses both the solutions found in practice, as well as research results from academia. The purpose of the conference is to bring the most recent findings and results in this field into the spotlight, and stimulate discussion between researchers, experienced professionals, and technology providers for software process improvement.

The conference programme includes top-notch keynote speakers, four half-day tutorials on various aspects of the conference theme, 38 high-quality presentations from 14 different countries, stimulating panel discussions, and two ESSI PIE dissemination sessions.

We wish to thank the BOOTSRAP Institute, the European Commission, the University of Oulu, and VTT Electronics for their support. We are also grateful to the programme committee, organizing committee, as well as to all the authors, speakers, and other contributors who made the quality of the conference so high.

Sincerely,

Markku Oivo	Pasi Kuvaja
VTT Electronics	University of Oulu

Conference Organization

General Chair:
Professor Dieter Rombach,
University of Kaiserslautern and Fraunhofer IESE
Germany

Program Co-Chairs:
Prof. Markku Oivo
VTT Electronics, Finland and
Fraunhofer IESE, Germany

Dr. Lawrence Votta
Brincos, Inc.
USA

Organizing Chair:
Assistant Professor Pasi Kuvaja, University of Oulu
Department of Information Processing Science
Finland

Local Arrangements Chair:
Mrs. Katja Salmela
VTT Electronics
Finland

Publicity Chair:
Dr. Adriana Bicego
Etnoteam S.p.A
Italy

Sponsors
BOOTRAP Institute
European Commission
University of Oulu
VTT Electronics

Programme Committee

Adriana Bicego, *Etnoteam, Italy*
Reidar Conradi, *NTNU, Norway*
Ilkka Haikala, *Tampere University of Technology, Finland*
Takeshi Hayama, *NTT Data Corp, Japan*
Bärbel Hörger, *Daimler-Benz, Germany*
Ross Jeffery, *University of New South Wales, Australia*
Erik Johansson, *Q-Labs, Sweden*
Karlheinz Kautz, *Copenhagen Business School, Denmark*
Marc Kellner, *SEI, Carnegie Mellon University, USA*
Munish Khurana, *Motorola, UK*
Kari Känsälä, *Nokia Research Center, Finland*
Pasi Kuvaja, *University of Oulu, Finland*
Risto Nevalainen, *STTF, Finland*
Harri Reiman, *Ericsson, Finland*
Günther Ruhe, *Fraunhofer IESE, Germany*
Veikko Seppänen, *VTT Electronics, Finland*
Rini van Solingen, *Schlumberger, The Netherlands*
Reijo Sulonen, *Helsinki University of Technology, Finland*
Ian Taylor, *Process Research Consultants, Austria*
Richard Veryard, *Veryard Projects, United Kingdom*
Otto Vinter, *Bruel & Kjaer, Denmark*
Giuseppe Visaggio, *University of Bari, Italy*
Claes Wohlin, *Lund University, Sweden*

In addition, the following persons have helped in reviewing the papers: Martin Höst, Magnus C. Ohlsson, Björn Regnell, Per Runeson, Anders Wesslen, Kari Alho, Casper Lassenius, Jyrki Kontio, Jarmo Hurri, Marjo Kauppinen, and Jari Vanhanen.

Contents

Keynote Address:

The Team Software Process

Speaker

Watts S. Humphrey

The Software Engineering Institute
Carnegie Mellon University, USA

The Software Engineering Institute has developed the Team Software Process (TSP)[SM] to help integrated engineering teams more effectively develop software-intensive products. This talk reviews the current problems with software development and shows how the TSP addresses them. For example, hardware-software projects in even very experienced groups generally have cost, schedule, and quality problems. Testing is generally expensive and time consuming, and often followed by many months of user testing before the products are fully usable.

The TSP shows engineering groups specifically how to apply integrated team concepts (IPPD) to the development of software-intensive systems. It walks teams and their management through a 3-day launch process that establishes goals, defines team roles, assess risks, and produces a comprehensive team plan. Following the launch, the TSP provides a defined and measured process framework for managing, tracking, and reporting on the team's work.

The TSP has been used with pure software teams and with mixed teams of 2 to 20 hardware and software engineers and it has been shown to sharply reduce the total cost of development and acquisition. TSP has been used for both new development and enhancement and with both commercial and imbedded real-

[SM] Team Software Process and TSP are service marks of Carnegie Mellon University.

time systems. A number of organizations are using the TSP and this talk describes some of their experiences. Finally, the talk briefly reviews the TSP introduction strategy and approach.

Keynote Address:

"In situ" Computer Aided Empirical Software Engineering

Koji Torii

Nara Institute of Science And Technology
8916-5, Takayama, Ikoma, Nara, JAPAN 630-0101

First I will describe what I mean by Computer Aided Empirical Software Engineering (CAESE), and its importance in (empirical) software engineering.Empirical software engineering may make general software engineering morevisible with hopefully quantitative knowledge. Empirical software engineering can be viewed as a series of actions used to acquire knowledge and a better understanding about some aspects of software development given a set of problem statements in the form of issues, questions or hypotheses. Our experience in conducting empirical software engineering from a variety of viewpoints for the last decade has made us aware of the criticality of integrating the various types of data that are collected and analyzed aswell as the criticality of integrating the various types of activities that take place such as experiment design and the experiment itself. Our experience has led us to develop a CAESE framework as a substrate for supporting the empirical software engineering lifecycle. CAESE supports empirical software engineering in the same manner that a CASE environment serves as a substrate for supporting the software development lifecycle.

Replication, one of the most important activities in empirical softwareengineering is not easy to do. Even in controlled experiments, detailinformation cannot be transferred. Some of the reasons for these difficulties are found in the difference of experiment environment, such as software tools, knowledge levels of subjects, and etc. Thus we propose CAESE, consisting of a process description just the same as for CASE.

The second point is "in situ" concept. In biological science, or any empirical studies, work domains in general are based on two different approaches: in vivo (naturalistic) setting, and in vitro (laboratory) settings. In vivo settings deal with real world problems by looking at software developers in the natural context in which the actual software development project is carried out. Work by Basili on TAME and the Software Experience Factory approach are a type of naturalistic setting studies. The Software Experience Factory provides mechanisms to trace an actual software development project, to accumulate and store "knowledge" that emerges during the project, and to make knowledge available and reusable for other projectsand project members.

In vitro studies, on the other hand, consist of controlled experiments. Originating from psychological studies, laboratory studies focus on a particular aspect of human behavior, trying to develop a generalized model of behavior, or focus on proposed techniques and methods to test their effectiveness. A study looks at a particular task, develops hypotheses about human behavior or techniques related to the task, and tests the hypotheses. The task in the study, for example, may concern defect detection, maintenance, or requirements inspection.

I hope my discussions may partially answer the following questions. Why isn't process improvement easy? Why isn't the transfer of the process improvement technology easy? Why isn't replication in the empirical software engineering done properly?

Panel:

Process Improvement versus Product Improvement

In the field of software engineering when we talk about improvements, we usually mean process improvements. We think in terms of controlling and improving the activities that are conducted during the development of software. But after all, who cares about processes: we want products. So what really matters to us is product improvements.

We have a clear notion and supporting models of how to achieve process improvements, but we lack more knowledge on how to achieve product improvements. So the questions to be discussed by this panel will be:

- Do better processes really lead to better products?
- How can we assess product quality in the wider sense?
- Can process measurements support product improvement?

The panellists have been selected from industry as well as research.

Otto Vinter (Chair),
Brüel & Kjaer Sound & Vibration Measurement, Denmark
Rini van Solingen,
Tokheim, and Eindhoven University of Technology, The Netherlands
Kari Känsälä,
Nokia Research Center, Finland
Bo Balstrup,
Danfoss Drives, Denmark
Terry Rout,
Griffith University, Australia

Panel: Position Statement 1

Otto Vinter

(ovinter@bk.dk)

Manager Software Technology and Process Improvement

Brüel & Kjaer Sound & Vibration Measurement

Denmark

Models for assessing and improving software processes have been around for some time now: ISO 9001, CMM, BOOTSTRAP, SPICE (ISO 15504). However, these models only implicitly and indirectly address the properties of the resulting products. The underlying assumption is: Improved processes will lead to improved products. And the correctness of this assumption has not been validated (yet?).

The results of software process improvements are typically:

- more precise estimates (better predictions on development time),
- quicker time-to-market (reduced time in rework),
- productivity gains (reduced development costs)
- fewer defects (and failures after release).

Focusing only on processes could lead to the development of products that will not fulfil the expectations of its users. It has been said that a company at the top CMM level 5 could still go bankrupt because it develops the wrong products.

Models that focus on assessing an organisation’s capability to improve its *products* are not so common. Product improvement activities aim at developing products with improved quality in a wider sense e.g. as perceived by the users. The underlying assumption is: Develop the right product rather than develop the product right. But do we know how to assess product quality in this wider sense in order to support improvements?

One model in this area is the ISO 9126 with its list of product quality attributes. These attributes are often referred to as the -ilities of a product:

- functionality,
- usability,
- reliability,
- maintainability,
- efficiency, and
- portability.

Another model for product improvement is found in the Concurrent Engineering field, e.g. Integrated Product Development (IPD). Product improvement activities can also be found in the Human Computer Interface (HCI) community and in the Requirements Engineering field e.g: user centered design, and modeling with scenarios. But these are limited to only part of the development life cycle, and are not normally regarded as a part of the software development activities. Furthermore, they only address a few of the ISO 9126 product quality attributes.

At Brüel & Kjaer, I have been working full time on process improvements for more than 6 years. I have had successes and failures. The failures have typically been in areas where the support from management was not present to back (enforce) the change of even clearly poor processes.

On one of my recent improvement projects (PRIDE) I discovered that the improvements we achieved on the product far exceeded the improvements on the process. The end product achieved much higher sales due to a better match of user needs and increased usability. And the new techniques spread much quicker throughout the organization than I have ever seen before. I (re)discovered that product improvements are of much higher interest to management than process improvements because they are business driven. The interest among developers is also higher because they are motivated by the end-result of their development efforts, rather than by the way development is performed.

Management normally does not see process improvements as having a concrete impact on the product and therefore on sales or bottom line. They are reluctant to support (enforce) process improvements which they cannot see an immediate benefit from. They want to look ahead for new opportunities and possibilities. And all I can show them is a rear view (what went wrong last time). Improving a process is looked upon as spending money on *removing problems* rather than *adding features*, which is counterintuitive to them.

I therefore started to look around for more comprehensive treatments of *product* improvements and found that this area was not well researched. We have a lot of knowledge on *process* improvement, but is there really a link between the two? Of course product development can only be achieved through a set of processes. But which processes lead to better products? And can it be measured?

Panel: Position Statement 2

Rini van Solingen

(R.v.Solingen@tm.tue.nl)

Tokheim, and Eindhoven University of Technology,

The Netherlands

The question whether better processes lead to better products can be used as a topic in a conference panel, simply because we apparently don’t know. This already indicates the underlying problem. We have so far not measured, or been able to measure, the relation between specific software processes and the resulting product.

Nobody questions whether there is a relation between process and product, because there is. Without a process (how chaotic it may be), there will be no product. So, this is not the issue. The issue is that we don’t know how to manage this process in such a way that the resulting product complies to our needs. Apparently there is a lack of knowledge on the relation between specific processes (read: methods, techniques or tools) and product quality. I use the term ‘quality’ for both functional and non-functional characteristics of a product.

Recent literature research carried out in a joined project of Delft and Eindhoven Universities of Technology, concluded that almost all literature in the software engineering field:

a. Does not indicate what impacts there are of new methods, techniques or tools on product quality. Many publications indicate that the method, technique or tool presented results in ‘better quality’, but what quality means in that context is rarely explained.

b. Does not contain any measurement result to support that the claim on ‘better quality’ is indeed valid.

This leads us to the conclusion that the relationships between process and product are unknown and should therefore be discovered in order to control product quality by a focus on the process.

Furthermore, this points to a second issue: what is product quality? Product quality will mean very different things in different contexts. If the process should result in the quality that is required, this needed quality will have to be specified. Without a specification of the product quality requirements it will be almost impossible to select a development process that will result in ‘good’ quality. However, product quality requirements are rarely specified in the software engineering field.

My position for this panel is that if we want to control software product quality with the process, we need:

- Detailed (and valid) models of impacts of processes on product quality, and

- A detailed (and measurable) specification of the product quality requirements.

Panel: Position Statement 3

Kari Känsälä

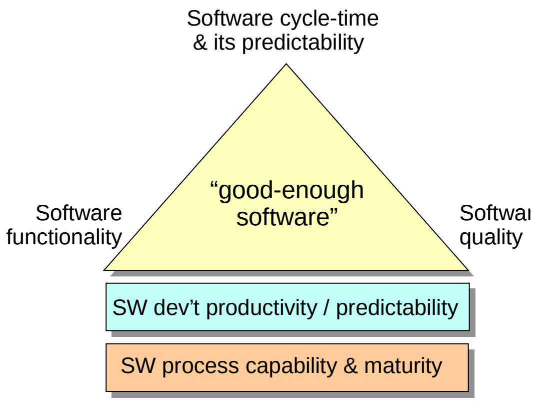
(kari.kansala@research.nokia.com)

Senior R&D Manager

Nokia Research Center

Finland

The Nokia Software Process Initiative (NSPI), that is operative in all Business Units developing software (SW) for Nokia's products, has derived all of its goals form the key business-related objectives of Business Units' R&D. The primary metrics of the NSPI (see Figure) are related to SW-intensive products:


Software cycle-time & its predictability
"good-enough software"
Software functionality
Softwar quality
SW dev't productivity / predictability
SW process capability & maturity

- SW cycle time (and its predictability)
- SW functionality
- SW quality

Productivity and process improvement metrics are regarded as secondary (see Figure), because achieving targets concerning those internal (R&D-related) metrics means only that the probability of achieving targets concerning primary external (customer-related) metrics gets higher.

The primary metrics form the "good-enough SW" triangle (named by Ed Yourdon), that should be regarded as the main framework concerning "product improvement". These metrics do not, however, cover the needs for product improvement metrics properly.

Nokia Research Center (NRC) has studied the ISO 9126 product quality characteristics as a promising solution for a more detailed product improvement framework. Two main difficulties have, according to our internal and literature findings, been identified in trying to implement ISO 9126 type of quality frameworks. Firstly, standard quality characteristics are hard to measure, because they do not necessarily match with recognized needs and possibilities of measurement. Secondly, they do not often match as such with the existing processes, methods, techniques and skills of current SW development community.

Instead of trying to push ISO 9126 type of frameworks as such, we in NRC have been mapping ISO 9126 like quality attributes to our own SW engineering R&D areas (e.g. user interfaces and usability, SW architectures, and SW process improvement) and our standard SW process frameworks (e.g. CMM, SPICE). Some examples can be given:

- work of Usability group matches directly with Usability characteristic of ISO 9126

- work of SW Architectures group matches directly or indirectly with the Functionality, Reliability, Efficiency, Maintainability and Portability characteristics of ISO 9126

- work of SW Process Improvement group matches indirectly via SW process frameworks; e.g. working on Requirements Management (a CMM Key Process Area) enhances all quality characteristics

This is our effort to perform "product improvement". If valid product quality metrics can be identified in conjunction with this work, they could be mapped to ISO 9126 characteristics.

Concerning the nature of Nokia’s product and solutions, it is extremely important to always recall the wider context: SW development is only a part of highly integrated product/solution development. Improved SW shows primarily via improved products and solutions, i.e. all SW improvement is not (always regarded as) product improvement.

Panel: Position Statement 4

Bo Balstrupo

(balstrup@danfoss.com)

Technology Manager

Danfoss Drives

Denmark

To discuss Product Improvement versus Process Improvement is like discussing the hen and the egg. Which comes first?

Normally process improvement is initiated by the need for improvement of product quality, which in my opinion is the right way to attack the problem. At least you have to go through your development process to find causes for the poor quality. For this CMM, BOOTSTRAP, SPICE and other assessment methods to help to find strengths and weaknesses.

Process improvement consists of three areas: Organisation, Methods, and Tools.

The most common error done by software developers is to focus on tools. Fools with tools are still Fools. To get it right focus must be on the Product Development organisation, not only on the Software Development organisation. The right organisation is the means to improved products and processes.

If the problem is developing the right product for the market, focus must be on requirements specification. A highly integrated effort with participants from the whole organisation will inevitably lead to improved products.

Improved specifications are not enough. You will still face the problem of extended time-to-market due to lack of good processes and lack of knowledge of the causes. Setting up a measurement program to track the development performance provide hints of where time eaters are. Based on this information a root cause analysis can pinpoint the problem areas.

Using a formal inspection or review method will on the first hand find many defects early in the project and secondly secure the vital product knowledge sharing and insight of good practises.

The final, but still very important method to improve, is testing. No, you can not test quality into your product, but with a planned and structured test effort you will, without doubt, get an improved product on the market.

Doing these few process improvements will surely lead to improved products on time. The time saved, not doing every thing twice, will release plenty of time to continue the improvement program for both products and processes.

Do not forget the power it releases in the organisation. Professional pride and commitment to perform will rise to the sky. And you might additionally get some satisfied customers.

My paradigm is: Develop the right product right.

Panel: Position Statement 5

Terry Rout

(t.rout@cit.gu.edu.au)

Manager of the Southern Asia Pacific Technical Center for SPICE

Griffith University

Australia

In my view it is false and dangerous to see the issue as a choice between process and product; one must keep focussed on both in order to achieve success. The techniques we have help us to diagnose and to set priorities, they do not force us to "choose".

SESSION 1:

PROFES: Methodology for Product Focused Process Improvement

A Validation Approach for Product-Focused Process Improvement

Andreas Birk[1] **Janne Järvinen**[2] **Rini van Solingen**[3]

[1]Fraunhofer IESE, Sauerwiesen 6, D-67661 Kaiserslautern, Germany, Andreas.Birk@iese.fhg.de

[2]VTT Electronics, P.O. Box 1100, FIN-90571 Oulu, Finland, Janne.Jarvinen@vtt.fi

[3]Tokheim , Industrieweg 5, NL-5531-AD Bladel, The Netherlands, R.v.Solingen@tm.tue.nl

Abstract

As the number of software engineering improvement methodologies and their adoption rate in industry increase, the validation of improvement methodologies becomes more and more important. Past validation studies show the effectiveness of improvement methodologies. However, they also reveal many technical difficulties for scientifically sound and detailed validation studies.

This paper surveys the state of the art in improvement methodology validation and derives recommendations for systematic validation studies, which are substantiated by experiences from the European PROFES project. PROFES has developed a product-focused software process improvement methodology and started its empirical validation already early in the project.

In brief, the main results of our validation work are: (1) Explicit product quality goals increase the effectiveness of process improvement and allow for causal analysis of observed improvements. (2) Validation should be based on explicit, a-priori set hypotheses involving multi-facetted validation criteria. (3) Improvement methodologies of different types, such as process assessment and goal-oriented measurement, are rather complementary than competing approaches.

1. Introduction

Validation of improvement methodologies is still a field of software engineering that is not very well elaborated yet. Several studies have been reported. However, they differ quite much with regard to their validation approaches, objects of study, and general validity of results. For this reason, we survey the approaches of past validation studies and provide a framework for validation of improvement methodologies. A recommended set of validation practices is derived. Their importance is underpinned by results from a replicated two-year case study in the European technology transfer project PROFES[1].

PROFES has developed an improvement methodology that is novel in multiple ways: (1) It focuses on process improvement that is driven by explicit product quality goals; therefore, explicit links are forged between aspects of the software process and their impact on the resulting software product quality. (2) The PROFES improvement approach integrates multiple improvement techniques that have, in the past, been applied in isolation, such as process assessments and goal-oriented measurement. (3) PROFES promotes a systematic and iterative approach of continuous improvement that is a-priori independent of any specific improvement technique and that follows the Quality Improvement Paradigm (QIP) / Experience Factory (EF) approach [1].

Methodology validation in PROFES is quite unique for multiple reasons: Empirical validation has been an explicit work task of methodology development from the beginning of the project. The validation study has been planned systematically prior to the start of methodology application. There have been close contacts between researchers and the methodology users allowing for close observation of the improvement programmes over a period of two years. Finally, the application projects have been actively involved in the definition of validation criteria as well as in the interpretation of the results.

This paper presents the basic validation strategy used in PROFES and illustrates it by selected validation cases. Section 0 surveys improvement methodologies

[1] ESPRIT project 23239, PROduct Focused improvement of Embedded Software processes. Supported by the Commission of the European Community. For further information about PROFES see: http://www.ele.vtt.fi/profes/

and their validation. It also lists requirements for improvement methodology validation. Section 0 briefly introduces the PROFES methodology for product-focused process improvement. The systematic approach to improvement methodology validation is introduced in Section 0. Section 0 summarises the main results and experiences from the methodology validation work in PROFES.

2. Improvement Methodologies and Their Validation

Improvement methodologies in software engineering build on quite a variety of different paradigms. Each paradigm has specific implications on methodology validation, determining relevant validation criteria and appropriate validation strategies. This section lists basic types of improvement paradigms that can be found in the literature[2]: Benchmarking, analysis, process modelling, learning, technology focus, and technology transfer. This classification serves as a baseline for surveying current work on methodology validation.

Benchmarking based improvement compares the software engineering processes of a project or organisation with a reference model of recommended software engineering processes. Improvement actions are identified based on the deviation between the assessed process and the reference model. *Process benchmarking* approaches are the Software Engineering Institute's (SEI) Capability Maturity Model (CMM) [23] [30], the emerging international standard ISO/IEC 15504 [25], and BOOTSTRAP [6]. *Quality management system benchmarking* approaches are the ISO/IEC 9000 standards family [24], the Malcolm Baldrige Award [8], and the European Quality Award [17].

Analytically based improvement applies software engineering measurement to investigate improvement potential, understand software development, identify improvement actions, and monitor their success (cf. the Goal/Question/Metric (GQM) approach [2] [35] [33] and other software measurement approaches [16] [29]). *Process modelling based improvement* establishes a core set of common

[2] Additional information from the literature survey can be found in [10].

software development practices within a project or organisation. Its main tool is software process modelling (cf.[31] [3]).

Learning based improvement focuses on the collection, organisation, and dissemination of good software engineering practice with a software organisation or from external sources into a software organisation (cf., EF/ QIP [1]). *Technology based improvement* places emphasis on a specific software engineering technology and fosters its dissemination and implementation throughout the software industry. Examples are software inspections (cf. [15]) and reuse (cf. [7] [27]). *Technology transfer based improvement* focuses on procedures and infrastructure for supporting the transfer and adoption of new technology. It does not emphasise or promote a particular software engineering technology (cf. [11]).

There exist multiple variations and combinations of these basic families of improvement methodologies. In general, two or more improvement methodologies should be combined for assuring the success of an improvement programme. For this reason, the PROFES improvement methodology integrates multiple techniques (see Section 0). However, little is know yet about the interaction and synergy between multiple improvement methodologies. Studying such interaction is an important objective of methodology validation. It requires further advanced, systematic validation approaches. PROFES has moved toward such approaches and collected experience about them (see Section 0).

It is important to note that none of the currently common improvement methodologies places much emphasis on setting explicit product quality goals for guiding improvement programmes. If an approach does so (e.g., the QIP), methodological support is limited. The PROFES methodology takes a step beyond that and fosters the role of product quality goals in improvement programmes (cf. Section 0).

The objective of validation is to demonstrate that an improvement methodology satisfies the expectations of its users. Validation strategies vary considerably across different types of improvement methodologies and the expectations of different user groups. Table 1 gives an overview of validation strategies that are reported in the literature. It is structured according to the basic types of improvement methodologies that have been described above. Each methodology can be analysed with regard to multiple different aspects (e.g., validity of

Improvement Methodology	**Validation Aspect**	**Typical Validation Approaches**
Benchmarking (Process Assessments)	Validity of process assessment methods as measurement instrument	Data collection: Survey, SW measurement Analysis: Correlational statistics Examples:[18][28][14]
Benchmarking (Process Assessment)	Effectiveness of process assessment as a means for improvement	Data collection: Survey, interview, SW measurement, subjective observation Analysis: Correlational statistics, descriptive statistics, qualitative analysis, subjective experience Examples:[18][4][36]
Benchmarking (Quality Management Systems Assessment)	Benefits from methodology application	Data collection: Survey, interview Analysis: Descriptive statistics, qualitative analysis Examples:[34]
Analytically Based	Benefits from methodology application	Data collection: Survey, interview, SW measurement, subjective observation Analysis: Descriptive statistics, qualitative analysis, subjective experience Examples: [9][12][20]
Learning Based	Benefits from methodology application	Data collection: Survey, interview, SW data, subjective observation Analysis: Descriptive statistics, qualitative analysis, subjective experience Examples: [9]
Technology Based	Benefits from technology application	Data collection: Interview, SW data, subjective observation Analysis: Correlational statistics, descriptive statistics, qualitative analysis, subjective experience Examples: [15][27]
Technology Transfer Based	Benefits from technology transfer strategy	Data collection: Interview, SW measurement, subjective observation Analysis: Descriptive statistics, qualitative analysis, subjective experience Examples: [19]

Table 1: Overview and classification of past improvement methodology validation work.

process assessments as a measurement instrument or their effectiveness in achieving a certain kind of improvement). Basic data can be collected using surveys, interviews, software measurement, and subjective observation of the researchers. Depending on the data available, various data analysis techniques can be applied: correlational statistics, descriptive statistics, qualitative analysis, and subjective interpretation based on personal experience. The literature review allows for the following conclusions:

- Improvement success is rarely measured in terms of specific product quality improvements. Most studies either report "overall quality improvements" (not stating quality attributes) or those that are not directly related to product quality (e.g., productivity or staff morale).
- Causal analysis has not yet been performed by any of the validation studies found in the literature. Hence, no study demonstrates that improvements are actually caused by the applied improvement methodology.
- Most studies use only a few validation criteria that vary across the studies. Hence, it is difficult to compare the results from these studies.
- Only a few studies investigate the effects of more than one improvement methodology. Interaction or synergy effects are hardly studied at all.
- Process assessments and some individual software engineering technologies (e.g., inspections and reuse) are by far better investigated than other improvement techniques.

Multiple authors have addressed principles of good methodology validation practice. Brodman and Johnson [4] found that for industrial organisations, cost of improvement are relevant mainly in terms of personnel effort, while government organisations focus on financial cost. Likewise, benefit from improvement, from an industrial perspective, is measured in terms of meeting (usually non-financial) organisational or customer goals. Simmons [32] underpins these findings. She emphasises that success should be evaluated using a set of multiple criteria, which can be derived from organisation-specific business drivers or organisational goals. El Emam and Briand [13] have found no validation study that establishes a causal relationship that improvement is the cause of benefits that are witnessed.

The PROFES methodology validation addresses these recommendations from past validation work, namely: (1) Multi-facetted evaluation criteria that have been derived from organisational goals and that consider also non-process factors. (2) Cost measured in terms of personnel effort. (3) Initial planning of the study to assure appropriate measurement and analysis as well as mitigation of bias. Based on the observations from our survey and the above recommendations we have identified a set of requirements on systematic validation of improvement methodology. These requirements have guided the PROFES methodology validation study. They are listed in Figure 1. It must be noted that methodology validation is a complex and challenging task. Meeting the stated criteria is very difficult. Most often, it is impossible to meet them all due to time and resource restrictions, or due to the inherent difficulties of in-vivo research.

(R1)	Define the improvement methodology that is subject of the study in a precise manner.
(R2)	Formulate hypotheses for validation explicitly at the beginning of the study.
(R3)	Ensure that the validation criteria are actually relevant from the relevant perspective.
(R4)	Ensure that the study uses a composite, multi-facetted validation criterion.
(R5)	Ensure that the validation criterion is measured consistently, in a uniform way across all participants of a study.
(R6)	Ensure that the validation criterion is measured using different, redundant indicators concurrently.
(R7)	Investigate causal relationships between the application of the improvement methodology and the validation criteria.
(R8)	Investigate possible moderating effects on validity of improvement methodology.
(R9)	For case studies: Establish close contacts between researchers and the studied software organisations.
(R10)	For surveys and experiments: Achieve representative samples.
(R11)	Ensure that the study is replicated in order to provide evidence from different sources or experimental settings.
(R12)	Identify possible bias of the investigation early. Take measures for mitigating it.

Figure 1: Requirements on systematic validation of improvement methodology.

3. Product-Focused Improvement

Most improvement methodologies in software engineering focus on software *process* improvement. Their underlying rationale is that improved software engineering processes result in better product quality (cf. [23]). Yet, none of these improvement methodologies makes the link between process and product quality explicit (cf. [30] [6] [24]). PROFES has developed a process improvement methodology [5] [22] [21] that explicitly considers product quality as starting point for improvement planning. Its main building blocks are:

- Improvement planning starts with the identification of the organisation's product quality goals. Process improvement actions are determined with regard to their expected impact on product quality.
- The link between product quality and the software engineering processes is established using a new kind of artefact called product/process dependency (PPD) model [5] [21]. A PPD repository deploys experiential knowledge about PPDs.
- Established improvement techniques such as process assessment (acc. to ISO 15504 [25]), goal-oriented measurement (acc. to GQM [2] [35]) are integrated into an overall framework to exploit their combined strengths.
- Improvement programmes are conducted in an iterative manner following the QIP/EF approach for continuous improvement [1].

The PROFES improvement methodology consists of six overall phases that are sub-divided into twelve steps for which established improvement techniques are suggested. Overall, the PROFES improvement methodology is modular and involves an upfront tailoring activity for adapting it to the specific needs and characteristics of the software organisation. It will be shown below that the explicit focus on product quality and the associated PPD concept facilitate methodology validation allow for detailed causal analysis of product quality achievements.

4. The PROFES Validation Approach

Validation of the PROFES improvement methodology started at the beginning of the PROFES project in early 1997. Based on the first blueprint of the methodology, the validation study was planned with early involvement of the methodology users. For a period of more than two years, the PROFES methodology has been applied yet in multiple projects at three industrial organisations: Dräger Medical Technology, Business Unit Monitoring (MT-M), Ericsson Finland, and Tokheim. During this time, the projects were subject to detailed observation by the researchers who are responsible for the validation work. The work was separated into two overall phases. Various kinds of information were collected about the improvement programmes in each phase. The PROFES methodology validation involves three basic types of validation criteria (i.e., multi-facetted validation):

- Achievement of product quality improvements through application of the PROFES improvement methodology (to be demonstrated by identifying causal links between methodology and product quality)
- Other benefit from applying the PROFES improvement methodology.
- Cost-effectiveness of the PROFES improvement methodology.

The following sub-sections briefly introduce study design and environment, and describe the three types of validation criteria used in PROFES.

4.1 Study Design and Environment

The basic design of the empirical work in PROFES is a twice repeated, three times replicated case study: The project is separates into two 15 months periods. In each period, the same kind of investigation is conducted at the three software organisations (Dräger MT-M, Ericsson Finland, and Tokheim). The PROFES methodology validation has been planned from the beginning of the project. GQM has been used to identify validation criteria. Two overall GQM goals were defined that differ in their viewpoints:

> Analyse the PROFES methodology with respect to cost/benefit for the purpose of characterisation from the viewpoint of the *methodology user* (Goal 1) / *methodology provider* (Goal 2) in the context of PROFES.

For each goal, questions and measures have been gained by interviewing representatives of the PROFES application projects or methodology developers, respectively. The results are defined in the form of two GQM plans, which have been used to plan data collection and analysis. Figure 2 outlines parts of their structure. It lists validation criteria and assumed impacting factors of the PROFES methodology validation.

Methodology User Viewpoint	*Methodology Provider Viewpoint*
Product and Process Improvements	**Product and Process Improvements**
Achievement of product quality goals	Product quality improvements
Standardisation of work practices	Process definition
Improvement of work practices	Process stability
Systematic Improvement	**Methodology characteristics**
Focused improvement actions	Customer viewpoint
Integrated business, product, and process issues	Quality and improvement awareness
Efficient management involvement	**Methodology definition and support**
Findings, Awareness, Understanding	Coverage of methodology (roles, phases, activities)
Knowledge about software and system	Guidance of methodology (processes, guidelines)
Awareness of software development capabilities	Documentation of methodology
Awareness of necessity of improvement	Tool support of methodology
Team Building & Organisational Culture	**Possible Impacting Factors**
Contribution to group synergy	Size of measurement programme
Awareness of necessity of improvement	Maturity of the software organisation
Possible Impacting Factors	Infrastructure of the software organisation
Maturity of the software organisation	Other ongoing improvement initiatives
Infrastructure of the software organisation	Organisational culture: Management commitment for the improvement programme
Other ongoing improvement initiatives	Degree at which quality improvement is integrated with regular software development activities
Project management's awareness of the improvement methodology	
Higher management's expectations on the improvement programme	

Figure 2: PROFES validation criteria and expected impacting factors.

Each of the three PROFES application projects develops some kind of embedded system. The individual products have quite different characteristics. Also the organisations and overall software development environments differ considerably from each other. Hence, it can be expected that the PROFES results have a quite high level of external validity. The design of the PROFES methodology validation and the systematic, up-front planning of the study satisfies widely the requirements (R1—6), (R8&9), and (R12) on systematic validation (see Section 0).

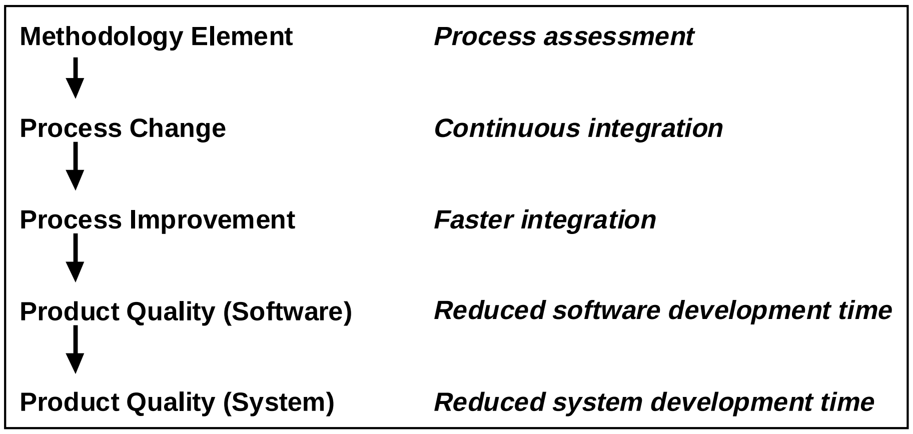


Figure 3: Pattern of the chain of evidence that is used for PROFES methodology validation and examples. (Note that the steps process improvement *and* product quality (software) *can be omitted in some cases.)*

4.2 Achievement of Product Quality Improvements

The core strategy of methodology validation in PROFES is to demonstrate that the application of the PROFES improvement methodology results in improvements of the developed system's product quality, where quality also involves aspects such as time-to-market and development costs. Validation should result in identifying and explicitly documenting a chain of evidence according to the pattern shown in Figure 3.

This should be done according to the following principles:

- Explicitly document the complete chain of evidence, listing each element in the chain.
- Clearly justify each transition in the chain from one element to the next.

- For each dependent element in the chain, thoroughly identify possible alternative explanations and try to refute them.
- Consider interaction effects of multiple methodology elements that were applied concurrently.
- Provide objective and quantitative evidence (based on measurement data) whenever possible.
- Provide subjective and qualitative evidence thoroughly.

These principles satisfy the requirement (R7) from Section 0 and are in accordance with basic principles of case study and qualitative research (cf. [26]).

Figure 4 shows an example validation case for achievement of product quality through application of the PROFES improvement methodology. It documents how the PPD concept of PROFES resulted in high usability of the final system product. Compared to the causal pattern from Figure 3, in the example case the steps *process improvement* and *product quality (software)* have been omitted, because the effects of the particular process change (i.e., incremental development) on process improvements (e.g., better manageability of tasks and work products) and on software product quality (e.g., defect density) are not critical to the system product quality (i.e., usability of the final product). For each causal relationship in the chain of evidence, possible alternative explanations are refuted and evidence for the causal relationship is provided. The validation case shows that the product/process dependency concept in the PROFES methodology is effective (*proof of existence*).

4.3 Multi-Facetted Benefit Criterion

Benefit from applying the PROFES improvement methodology is measured using multiple facets of benefit. Hence, the PROFES methodology validation complies with requirement (R4) in Section 0. The different facets of benefit are listed in Figure 2. They show that industrial software organisations—the users of the PROFES improvement methodology—want to see (1) that product improvements can be achieved (see also Section 0), (2) that various kinds of process improvements happen, (3) that the improvement programmes show certain characteristics (e.g., tailorability and efficient use of management resources), (4) that knowledge and awareness of multiple software development aspects increase, and (5) that team building and organisational culture are supported. The methodology provider viewpoint adds further validation criteria such as the quality of the methodology's documentation. Example early benefits from GQM as part of the PROFES improvement methodology are:

- *Enhanced definitions of software development processes:* Already in the planning phase of the GQM measurement programmes, a need for more detailed or updated software process definitions showed up.
- *Knowledge about software and system:* GQM measurement has widened the project teams' knowledge about software and system, resulting in even better informed technical work and decision making.
- *Fine-tuning of improvement actions:* During the GQM feedback sessions, previous improvement actions were fine-tuned by the software engineers in order to improve their efficacy.

Similar information has been gained concerning later stages of the GQM measurement programmes, ISO 15504 process assessments, software engineering experience management, and other parts of the PROFES methodology. A presentation of the detailed results from the benefits investigation would exceed the scope of this paper. The results provide a detailed view on the application of the PROFES improvement methodology.

PROFES Methodology Validation Case	
Methodology Element	**Product/process dependency (PPD)**
Process Change (note: this process change affects the system product quality directly. Intermediate affects on process and software quality are not really relevant and do not need to be investigated.)	**Incremental development** (Six months cycles from requirements to system test. Implementation of core functionality in early increments in order to test them early and multiple times.)
	Evidence for implementation of change: The project schedule substantiates cycle duration. Both product increments achieved until today were fully operational for conducting user test; hence, the main functionality was present.
	Causal relationship: Usage of PPDs during project planning was emphasised by the PROFES project. Project management cared especially much for the identification of product quality goals and identified measures that were likely to help achieving the goals. In this case, quality and development time goals, together with the fact that many aspects of the project would be new to the team and the organisation resulted in the decision of an incremental development process.
	Possible alternative explanations: 1. Change due to process assessment 2. Not actually a change but standard development practice
	Refutation of alternative explanations: ad 1.: First process assessment took place after the decision for incremental development was taken. ad 2.: None of the previous projects did address incremental development.
System Product Quality	**Usability of the product is good** • Layouts of screen and control devices are attractive and user-friendly • Handling of the product is user-friendly
	Evidence for achievement of the software product quality: Usability of the first product increment was not satisfactory. The second increment shows good usability (according to similar statements from product marketing and from the development team). Engineers report that there was enough time for evaluating product usability and for improving it after evaluation.
	Causal relationship: Incremental development resulted in a good user interface.
	Possible alternative explanations: 1. Usability requirements were easy to implement 2. Usability only due to new hardware features
	Refutation of alternative explanations: ad 1.: The product is of a totally new type, new user interface hardware was used, the user interface is much more complex than with previous products. ad 2.: The first increment's usability with the same hardware was not fully satisfactory.

Figure 4: Example case for validation of an element of the PROFES improvement methodology.

4.4 Cost-Effectiveness of the Improvement Methodology

The third type of methodology validation criteria in PROFES is cost-effectiveness. The GQM interviews for planning the evaluation work have resulted in the following facets of cost-effectiveness:

- Overall effort for the improvement programme
- Effort for the improvement programme by key personnel: Managers, software engineers, improvement team, and external consultants.
- Tailoring effort for the improvement methodology when setting up the improvement programme.

The related measurements have provided detailed effort data about the execution of BOOTSTRAP process assessments and GQM measurement programmes. It involves the number of hours spent by each participant of the improvement programme for each activity of the respective method. Table 2 shows an example effort model for one variant of BOOTSTRAP assessments.

During the second phase of PROFES, process assessments and measurement-related activities were conducted in an integrated manner. The effort data from these activities allows for investigation of possible synergy effects between the two techniques.

Activity	Role					Total
	Lead Assessor	Assessor	Manager	Engineer	Facilitator	(Effort in pers. hrs.)
Preparation	18	20			2	40
Opening Briefing	0.5	0.5	2	1	0.5	4.5
Assessment SPU	7.5	7.5	2.5		1	18.5
Assessment Project	27	26	4.5	4	3	64.5
Evaluation	32	16				48
Review	10	10				20
Final Meeting	7	7	4	4	6	28
Report Preparation	44	4				48
Report Review	2	8				10
Total	148	99	13	9	12.5	281.5

Table 2: Example effort model of BOOTSTRAP process assessments.

5. Conclusions

A survey of literature on validation of improvement methodology has uncovered that causal analysis of the assumed effects of improvement methodologies have hardly been conducted yet. Furthermore, most validation studies apply only a very few validation criteria that are rarely derived from the users of improvement methodologies.

In the context of ESPRIT project PROFES, we have had the opportunity to conduct methodology validation in a manner that is in multiple respects different from past methodology validation work:

- We have investigated causal relationships between improvement methodology and product quality improvements.
- A multi-faceted validation criterion has been derived from the users of the improvement methodology; it covers multiple aspects of benefit and cost (measured in terms of effort) of methodology application.
- We were able to investigate interaction and synergy effects between different improvement techniques (e.g., process assessment and software engineering measurement) as well as the impact that an explicit product quality focus has on software process improvement.

The setting in which the PROFES methodology validation could be performed has been quite unique. First, empirical validation has been an explicit work task of methodology development and started very early in the project. Second, the close contacts between the validation team and the application projects allowed for very detailed observation of the methodology application over a considerably long time period. Third, the PROFES software projects were not only delivering data but also participating actively in the definition of validation criteria as well as in the interpretation of the results.

The objective of this paper is to explore methodological aspects of the validation of improvement methodology. A detailed presentation of results does not fit into the scope of this paper. However, without providing detailed evidence in this context, we can report the following main findings about product-focused process improvement:

- An explicit product quality focus can have much impact on effectiveness and efficiency of improvement programmes. It guides decisions that can be difficult to justify through process assessments or software engineering measurement alone, and it ensures that other improvement techniques can be applied in a more effective manner. For instance, PPD models can focus a process assessment to those processes that are expected most critical for achieving a required product quality.
- Many synergy effects can be derived from the informed combination of common improvement techniques, such as process assessments, process modelling, and measurement. For instance, detailed modelling of selected processes facilitates process assessments and measurement programmes.
- The two improvement techniques that are currently most common, namely software process assessments and software engineering measurement, are cost-effective.

Based on our findings, we can argue that product-focused improvement using the principles of the PROFES improvement methodology is more effective than the non-integrated application of improvement techniques that consider product quality goals only in an implicit manner. We also encourage the reuse of our validation approach in future improvement programmes. It can facilitate customisation and management of improvement programmes and also provide further evidence for the validity of product-focused improvement from which other software organisations and software engineering research can benefit in the future.

6. Acknowledgements

The authors would like to thank all members of the PROFES consortium for many fruitful discussions. Special thanks go to the application projects at Dräger MT-M, Ericsson Finland, and Tokheim. We would also like to thank Dirk Hamann and Dietmar Pfahl for reviewing an earlier version of this paper and Felix Kröschel for the very valuable help during data collection and analysis.

The Commission of the European Community has supported this work through ESPRIT project no. 23239 PROFES.

References

[1] V.R. Basili, G. Caldiera, and H.D. Rombach. Experience Factory. In J.J. Marciniak, ed., Encycl. of SE, vol. 1, pp. 469–476. John Wiley & Sons, 1994.

[2] V.R. Basili, G. Caldiera, and H.D. Rombach. Goal Question Metric Paradigm. In J.J. Marciniak, ed., Encycl. of SE, vol. 1, pp. 528–532. John Wiley & Sons, 1994.

[3] A. Bröckers, Ch. Differding, and G. Threin. The role of software process modeling in planning industrial measurement programs. In Proc. of the 3rd Int. SW Metrics Symp., Berlin, March 1996. IEEE CS Press.

[4] J.G. Brodman and D.L. Johnson. Return on investment (ROI) from software process improvement as measured by US industry. Software Process–Improvement and Practice, 1(1):35–47, Aug. 1995.

[5] A. Birk, J. Järvinen, S. Komi-Sirviö, M. Oivo, D. Pfahl, PROFES – A Product-driven Process Improvement Methodology, In Proc. of the 4th Europ. SW Proc. Impr. Conf. (SPI '98), Monte Carlo, Monaco, Dec. 1998.

[6] A. Bicego, M. Khurana, and P. Kuvaja. BOOTSTRAP 3.0 – Software Process Assessment Methodology. In Proc. of the SQM '98, 1998.

[7] T.J. Biggerstaff and A.J. Perlis. Software Reusability – Applications and Experience, vol.s I & II. ACM Press, 1989.

[8] M.G. Brown. The Baldrige criteria - better, tougher and clearer for 1992. Journal for Quality and Participation, 15(2):70–75, March 1992.

[9] V. Basili, M. Zelkowitz, F. McGarry, J. Page, S. Waligora, and R. Pajerski. SEL's sw process-improvement program. IEEE SW, 12(6):83–87, Nov. 1995.

[10] A. Birk, J. Järvinen, and R. van Solingen. A validation approach for product-focused process improvement. Fraunhofer IESE Technical Report IESE-005.99. Kaiserslautern, Germany. 1999.

[11] J. Christian, M. Edward, S. Redwine, and L. Tornatzky. Using new technologies. Technical Report SPC-92046-CMC, SW Prod. Cons., 1993.

[12] M.K. Daskalantonakis. A practical view of software measurement and implementation experiences within Motorola. IEEE Trans. Software Eng., Vol. 18, No. 1, 1992, pp. 998—1010.

[13] K. El Emam and L. Briand. Costs and benefits of software process improvement. In C. Tully and R. Messnarz, eds., Better Software Practice for Business Benefit. Wiley, 1997.

[14] K. El Emam and A. Birk. Validating the ISO-IEC 15504 measure of software requirements analysis process capability. ISERN Technical Report ISERN-99-02. Kaiserslautern, Germany. 1999.

[15] M.E. Fagan. Advances in software inspections. IEEE Transactions on Software Engineering, 12(7):744–751, July 1986.

[16] W.A. Florac, R.E. Park, and A.D. Carleton. Practical software measurement: Measuring for process management and improvement. Technical Report CMU/SEI-97-HB-003, SEI, Carnegie Mellon University, April 1997.

[17] European Foundation for Quality Management. Guidelines on self-assessment. brochure.

[18] D. Goldenson and J. Herbsleb. After the appraisal: A systematic survey of process improvement, its benefits, and factors that influence success. Technical Report CMU/SEI-95-TR-009, ESC-TR-95-009, SEI, Carnegie Mellon University, Aug. 1995.

[19] R.B. Grady and T. van Slack. Key lessons in achieving widespread inspection use. IEEE SW, 11(4):46–57, July 1994.

[20] C. Gresse, B. Hoisl, H.D. Rombach and G. Ruhe. Kosten-Nutzen-Analyse von GQM-basiertem Messen und Bewerten - Eine replizierte Fallstudie. In O. Grün/L.J. Heinrich, eds., Wirtschaftsinformatik: Ergebnisse empirischer Forschung, pp. 119–135. Springer, Wien; New York, 1997.

[21] D. Hamann, J. Järvinen, A. Birk, D. Pfahl. A Product-Process Dependency Definition Method. In Proc. of the 24th EUROMICRO Conf.: Workshop on SW Process and Product Impr. pp. 898-904. IEEE Computer Society Press, Västerås, Sweden, Aug. 1998.

[22] D. Hamann, J. Järvinen, M. Oivo, D. Pfahl. Experience with explicit modelling of relationships between process and product quality. In Proc. of the 4th European SW Process Improvement Conf., Monte Carlo, Monaco, Dec. 1998.

[23] W.S. Humphrey. Managing the Software Process. Addison Wesley, Reading, Massachusetts, 1989.

[24] International Organization for Standardization. ISO 9000: Quality management and quality assurance standards; Part 3: Guidelines for the application of ISO 9001 to the development, supply and maintenance of software. Geneva, Switzerland, 1991.

[25] ISO/IEC: Information Technology: Software Process Assessment. ISO/IEC Tech. Rep. 15504 Type 2, ISO (Ed.), Geneva, Switzerland, 1998.

[26] A.S. Lee. A scientific methodology for MIS case studies. MIS Quarterly, March 1989, pp. 33—50.

[27] W.C. Lim. Effects of reuse on quality, productivity, and economics. IEEE SW, 11(5):23–30, Sep. 1994.

[28] F. McGarry, S. Burke, and B. Decker. Measuring the impacts individual process maturity attributes have on software products. In Proc. of the 5th Int'l SW Metrics Symposium, pp. 52—62. IEEE Computer Society Press, Nov. 1998.

[29] NASA. Software measurement guidebook. Technical Report SEL-84-101, NASA Goddard Space Flight Center, Greenbelt MD 20771, July 1994.

[30] M.C. Paulk, B. Curtis, M.B. Chrissis, and C.V. Weber. Capability maturity model, version 1.1. IEEE SW, 10(4):18–27, July 1993.

[31] D.E. Perry, N.A. Staudenmayer, and L.G. Votta. People, organizations, and process improvement. IEEE SW, 11(4):36-45, July 1994.

[32] P. Simmons. Quality outcomes: Determining business value. IEEE SW, Jan. 1996.

[33] R. van Solingen and E. Berghout. The Goal/Question/Metric method: A practical handguide for quality improvement of software development. McGraw-Hill, 1999.

[34] D. Stelzer, M. Reibnitz, and W. Mellis. Benefits and prerequisites of iso 9000 based software quality management. SW Process Newsletter, (12), 1998.

[35] F. van Latum, R. van Solingen, M. Oivo, B. Hoisl, D. Rombach, and G. Ruhe. Adopting GQM-Based Measurement in an Industrial Environment. IEEE SW, 15(1):78–86, January 1998.

[36] H. Wohlwend and S. Rosenbaum. Software improvements in an international company. In Proc. of the 15th Int'l Conf. on SE, pp. 212–220. IEEE Computer Society Press, May 1993.

Establishing continuous assessment using measurements

Janne Järvinen[3], VTT Electronics, Finland & Fraunhofer IESE, Germany
Rini van Solingen, Tokheim & Eindhoven University of Technology, The Netherlands

Abstract

Software process assessments have become commonplace in the software industry. Assessments are sometimes regarded, however, as too infrequent, expensive and disruptive. Hence, there is a clear need for alternative ways to assess the current status of software processes and monitor the implementation of improvement activities. An assessment of software processes is based on finding indicators for establishing whether certain processes exist and how well they are performed. Based on the assessment outcome, improvement actions will be identified and guided with appropriate measurements. The software process (re-) assessment can be supported and made more frequent by using data from these measurements, and thus reducing the cost of the assessment.

In the European project PROFES (PROduct-Focused Improvement of Embedded Software processes) an integrated, product-focused process improvement methodology has been developed and tested in the industrial partner sites. Among other things, PROFES integrates process assessment and software measurement to enable continuous assessment.

This paper shows how this combination of assessment and measurement was done in practice. It also describes examples of measurements and their connection to the assessment method.

[3] Contact the authors via: janne.jarvinen@vtt.fi or jarvinen@iese.fhg.de

1. Introduction

Many companies that make investments for software process improvement could use their resources much more efficiently than today. The issue is that software process assessment and software measurement programmes are mostly applied separately, while there is some overlap on measuring processes. A single metric can be used for multiple purposes. A metric can tell about a defect found in a design document inspection, but it can also tell about the inspection process efficiency. Moreover, the same metric can give information for establishing whether an inspection process is performed in a given project. This kind of measurement information can also be used to facilitate software process assessment. The a priori assumption of this paper is that the cost of a software process assessment is reduced if parts of the interviews and data gathering can be integrated.

The paper is structured as follows: In Chapter 0 the relevant background for continuous assessment and goal oriented measurement is introduced. Chapter 0 describes the fundamental principles and expected benefits of continuous assessment. Chapter 0 portrays the PROFES project and the setting for this study. Chapter 0 contains a description of the practical steps taken in the PROFES project to apply continuous assessment. Also, an example is included to illustrate the links between measurement and assessment.

2. Background material

2.1 Bootstrap

The BOOTSTRAP methodology [3] is an ISO 15504 (SPICE) compliant software process assessment method that also supports the ISO-9001 standard. Process assessment is performed both at organisational level (SPU-Software Producing Unit) and at project level. At the organisational level assessment the goal is mainly to assess the written official processes and in project level assessment the goal is to assess how these processes are executed in practice. With process assessment the strengths and the weaknesses of current processes are identified through comparison with the assessment reference model.

The BOOTSTRAP methodology has been originally developed for the needs of improving the capability the European software intensive industry. BOOTSTRAP was designed to cover ISO 9001 requirements and to fit also to small and medium sized organisations. During PROFES project the BOOTSTRAP methodology has been enhanced to fulfill the requirements stated for embedded systems development through the extension with new embedded software specific process areas.

2.2 GQM

One of the most popular methods for software measurement is the Goal/Question/Metrics approach (GQM) [1] [10], depicted in Figure 1.

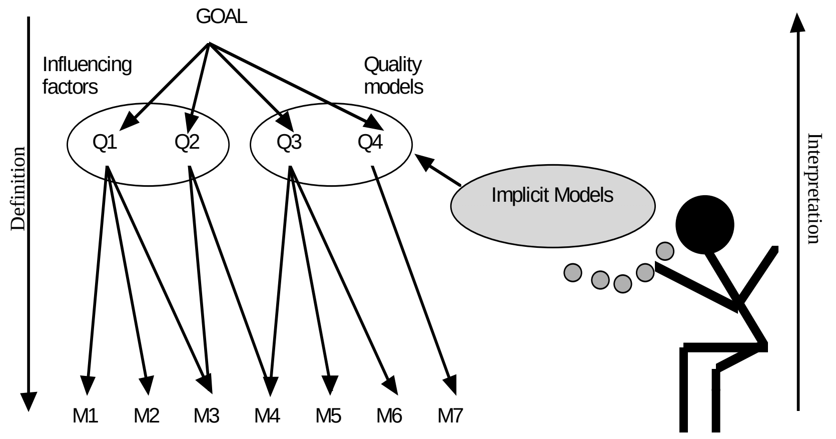


Figure 1: The Goal/Question/Metric paradigm.

GQM represents a systematic approach to tailoring and integrating goals with models of the software processes, software products, and with particular quality perspectives of interest. GQM focuses on the specific needs of the software project and of the development organisation. Measurement goals are defined on the basis of high-level corporate goals, and refined into metrics. In other words, GQM defines a certain goal, refines this goal into questions, and defines metrics that must provide the information to answer these questions. The GQM paradigm provides a method for top-down metric *definition* and bottom-up data *interpretation*. GQM is goal-oriented, which makes it especially popular in goal-driven business environments.

The principles of GQM measurement are:

- A measurement programme must reflect interests of data providers and must be based on the knowledge of real experts on the measurement goals. In this paper these are members of the software *project team*.
- Since the design of the measurement programme is based on the knowledge of the project team, only they can give valid interpretations of the collected data. Therefore, they are the only ones allowed to interpret data.
- Due to the limited amount of time of project members, and their commitments to project planning, conflicts of interest may occur when all improvement efforts are assigned to the project team. Therefore a separate team, a *GQM team*, must be created that facilitates the collection and analysis of measurement data by performing all operational activities not necessarily to be executed by the project team.

These principles imply that the members of the GQM team offer a service to the software project team by doing most of the technical work, related to setting up and performing the measurement programme. Essentially, during execution of the measurement programme, the GQM team provides a data validation and analysis service, by organising 'feedback sessions' in which graphical measurement data is presented to the project teams [9] [10].

3. Principles of continuous assessment

Software process assessments have become an established part of improvement programmes in the software industry (cf. e.g. [8]). However, the way in which assessments are performed has been characterised as too infrequent, expensive and disruptive to the software engineers (cf. e.g. [4], [11]). Hence, there is a clear need for alternative ways to assess the current status of software processes and monitor the implementation of improvement activities.

3.1 Principles of continuous assessment

Typically, an assessment is an annual or biannual snapshot of the software development activities, and is conducted as a self-assessment or using an external assessment team. Information gathering is done manually through document reviews and interviews. Use of supporting tools is minimal.

The basic idea of continuous software process assessment is to collect relevant information from the software process, as it becomes available. This information can then be consolidated and used to help an assessor in judging the process status.

There is a paradigm shift with continuous assessment. Information is continuously gathered using existing data from the development process where possible. While a competent assessor or team using available information does the act of assessment in a traditional sense, the continual manner how the assessment is done changes the role of assessment within process improvement.

The degree of continuity and automation determines how embedded the assessment is in the software development process. If majority of assessment information is gathered (automatically) via a measurement programme, the notion of Measurement bAsed Assessment (MAA) clarifies this special instance of continuous assessment. In this article we assume a hybrid approach for continuous assessment where part of the data for assessment purposes are collected automatically and part of data is collected manually via questionnaires and checklists. Given the process and technological capability of the organisations participating in the exploratory study at hand the hybrid approach has been considered optimal.

The ISO 15504 is used as a reference framework for the software process capability. When the ISO 15504 reference model is enhanced with the assessment model defined in part 5 of the SPICE it is possible to find links between measurable objects and the ISO 15504 framework (cf. Figure 2). Specifically, the *assessment indicators* provide the adequate detail for connecting process information with the framework. The indicators of *process performance* are used to determine whether a process exists in practice.

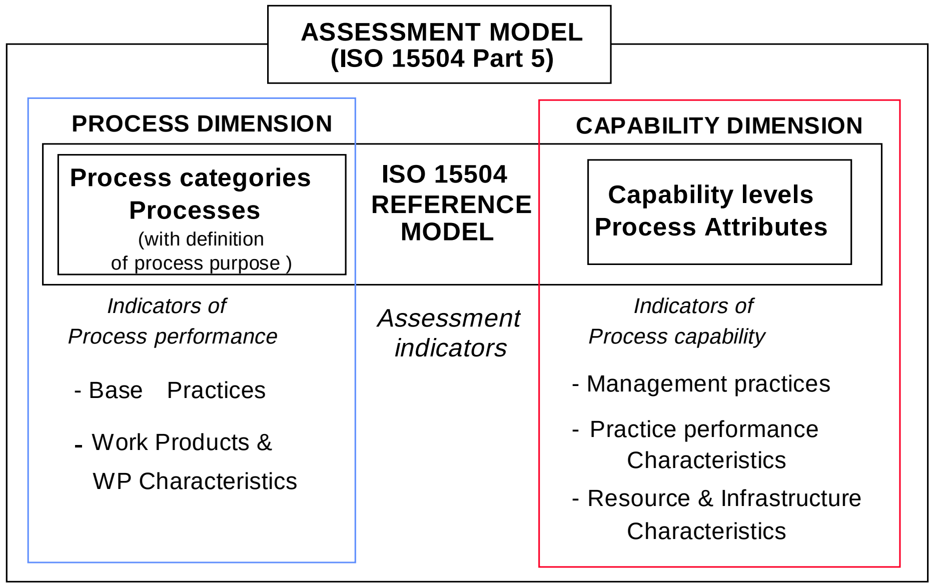


Figure 2: The ISO 15504 framework.

For example, the software design process (cf. ENG.1.3 in ISO 15504 reference model, [5]) is considered as existing if it can be determined that there exist documents that specify

- an architectural design that describes the major software components that will implement the software requirements;
- internal and external interfaces of each software component;
- a detailed design that describes software units that can be built and tested;
- consistency between software requirements and software designs.

If a software design process is functioning in an organisation it should be fairly straightforward to determine the existence of the documents that satisfy the goals listed above. This information could be contained, e.g. in a document management system that keeps track of the documents produced against a specified process. A report from this system would then help the assessor in determining whether the software design process is performed.

Further, the ISO 15504 indicators of process capability are used to determine how capable an existing process is. Linking information from the measurement system to the management practices, characteristics of practice performance, resource and infrastructure can assist an assessor in determining how well the process is performed as intended by ISO 15504. For example, the performance management attribute 2.1 of SPICE level 2 can be considered as fulfilled if

- objectives for the performance of the process will be identified (for example, time-scale, cycle time and resource usage);
- the responsibility and authority for developing the work products of the process will be assigned;
- the performance of the process will be managed to produce work products that meet the defined objectives.

3.2 Expected benefits

There are two main areas where continuous assessment is expected to bring benefits over the traditional approaches:

- Process visibility
- Assessment cost

With continuous assessment the process implementation becomes more visible. This means that it is possible to see in detail what is done in the software process. For example, this enables close observation of improvement activities so it is more apparent whether new practices are adopted and successful long before the usual re-assessment. Continuous assessment also provides the means to detect process deviations earlier thus helping to manage process implementation in two ways: Firstly, giving early signals on practices that are not being adopted, indicating that people should be supported with the process adaptation. Secondly, suggesting potentials for process change. Typically, defined processes and procedures are quite rigid. In practice, processes are dynamic, so they are always trying to change. Having support for visualisation of process implementation can help in identifying processes that should change or are already being changed in practice by the people using the process. This way processes can be living representations of the work rather than folders on bookshelves collecting dust.

The assessment costs are expected to be reduced with continuous assessment. The working hypothesis is that collecting information from the software process as it becomes available reduces the time needed for the interviews and document analysis during an assessment. This data collection can greatly be supported by appropriate tooling (cf. [7]). The key is to integrate the data collection into the work processes in such a way that it is a natural part of the work. This can be achieved in two ways: Either the data collection is essential for the work to be performed (e.g. writing an inspection report) or that the work automatically leaves marks in the tools and databases of the company. In integrating and automating the data collection the cost/benefit –ratio should be regarded to find an optimal mix.

4. Study Environment and Participants

The work presented in this paper on continuous assessment, results from the Esprit project PROFES (23239). In the PROFES project, an integration is made between assessments and measurement programmes in order to increase their efficiency and effectiveness.

Two roles were present in the PROFES project for the development of continuous assessment: method provider, and application providers. The method providers were: VTT Electronics (Finland), University of Oulu (Finland), Fraunhofer IESE (Germany), and Etnoteam (Italy). These four partners have sound experiences with both assessments and GQM measurement programmes resulting from international cooperation projects such as BOOTSTRAP, SPICE, TAME, ESSI/CEMP, and PERFECT. The application providers were the industrial companies offering practical expertise and real development projects for PROFES. These three industrial companies were: Dräger Medical Technology (The Netherlands), Ericsson (Finland), and Tokheim (The Netherlands). All of these companies are active in the development, maintenance and services of systems but in three different markets: medical technology, telecommunications, and petroleum retailing. The application providers participated by testing the continuous assessment methods in some of their projects and/or departments.

The main objective to start with the development of continuous assessment was the critique in the companies on the cost and duration of an assessment. Since assessments were used in the PROFES methodology for process baselining only, the required effort was considered too high. Also the calendar time (two months) between the start of an assessment and the feedback of the findings was considered to be much too long. On the other hand a measurement feedback procedure was used that provided feedback on much shorter intervals with considerably low effort to produce the results. However, the measurements conducted and analysed were applicable for assessments directly. This observation has supported one of the objectives set for the project, i.e. to integrate the PROFES assessments with the PROFES measurement activities. Initially, a full assessment was carried out in the three application providers. The critique on the assessment cost and duration came from this initial assessment.

The second round of assessments had the objective to overcome this critique. Two solutions were found:

- Limit the assessment to a sub-set of processes. Not all processes were assessed, only those that have been worked on to achieve improvements were re-assessed. This reduces the cost and shortens duration. However, still a first large assessment is required.
- Use measurement data on the improved processes as evidence of improvement. If the measurement data collected for the active measurement programmes had an overlap with assessment indicators, this data was used as evidence in the assessment.

This second round of assessments indicated that it is feasible to use measurement data to shorten assessment duration and decrease assessment cost. However, it also appeared that several of the assessment indicators were not covered by the available measurement data. Therefore, additional effort was needed to collect the evidence for these indicators.

Based on the experiences in the second round of assessments, it was concluded that a more thorough integration of a measurement programme with assessment indicators is needed in order to really carry out continuous assessment. Translating the assessment indicators to software metrics, and applying this list as a checklist during the design of a measurement programme, appears to be a way forward to continuous assessment.

The experiences in the industrial companies were the main starting point for developing the concept of continuous assessment, and the industrial companies also specified most requirements for continuous assessment. Currently, the concept of continuous assessment is being further detailed and developed. In parallel, the companies are experimenting with the draft concepts for continuous assessment in order to acquire experience with it and adjust these concepts to make them as practical as possible.

5. Continuous Assessment Approach in PROFES

In PROFES, we decided to investigate the continuous assessment by combining the BOOTSTRAP assessment approach with GQM–based measurement. The approach has been motivated and constrained by the needs of the industrial cases aiming to explore the practical applicability of continuous assessment. This chapter describes the method used within PROFES for continuous assessment and gives an example of the method application.

5.1 Steps for applying Continuous Assessment

The proposed steps to apply continuous assessment are as follows:

I. Select processes

II. Define indicators for process dimension

III. Define indicators for capability dimension

IV. Update GQM and measurement plans

V. Collect data and analyse results

I. Select processes

The principle in selecting processes for continuous assessment in PROFES has been that only those processes are included that are either critical or currently being improved. Generally, it is good to start small and gain experience with continuous assessment. In short, good candidates are those that are a) already measured, b) being or planned to be improved, and c) extensively supported by tools (to minimise manual data collection). The selected processes should be reviewed so that

- Target rating is recorded for each practice – can be the same as current rating if only monitoring is attempted. This is the starting point to systematically govern the (improvement) activities.
- Applicable sources for measurement data are defined. Examples of good data sources with potential for data collection automation are Lotus Notes, MS-Project, any Configuration Management system, any database that is used to collect project data [cf. 7]. However, the data does not have to be always automatically collectable (although this is usually preferred).

For each process the most important metrics are those indicating whether the process is performing or not, i.e. producing useful results and fulfilling the purpose of the process. This is the ISO15504 process dimension. Depending on the chosen scope the measurements for ISO15504 capability dimension can also be regarded. These measurements relate to the control, management, and improvement aspects of the process. See later in this chapter for an example. Note that there may be practices that are better left for assessment interviews; i.e. not everything needs to be covered automatically.

II. Define indicators for process dimension

In ISO15504 process dimension there are Base Practices, that are the minimum set of practices necessary to perform a process successfully. For example, the base practices for the Software Construction Process (ENG.1.4) are: Develop software units, Develop unit verification procedures, Verify the software units, Establish traceability, i.e. typically covering coding and unit testing in a software life cycle. Suitable metrics for base practices are usually those that give evidence of the base practice existence, i.e. that something has been done that contributes to fulfilling the purpose of the process. Mostly, the information should be found from the artefacts that are produced in a particular process, i.e. work products.

III. Define indicators for capability dimension

The ISO15504 capability dimension should also be examined for the selected processes. Basically, the capability dimension contains information on how well the practices are done, and how well the process runs. Usually, going through Level 2 of the capability dimension is enough as this is the state of the practice today. Recent SPICE assessment trials results show that only 12% of process instances (total 341) were higher than Level 2 [11] . Naturally, higher levels can be revisited depending on target capability. The information for the capability dimension can mostly be found from the project plan, project reporting documents, configuration management system, and the actual work products.

IV. Update GQM and measurement plans

The definition of relevant measurements for continuous assessment does not necessarily include a GQM goal tree formulation, as the ISO15504 processes form the structure for the investigation. However, an existing GQM plan is an excellent source of information. Some of the GQM measurements may also be used to facilitate software process assessment. Augmenting an existing measurement programme with process capability focus can bring added value for reasonable costs. For example, it is possible to monitor process improvement activities closely and evaluate the effectiveness of the process changes.

The integration of measurement activities into the software process must be planned with care. Usually this involves at least minor changes to the process as data must be recorded or structured in a way that is processable later. Software tools and databases are a key source for process data but even there effort is needed to structure, convert and extract data from various tools and databases. Some data may also be entered manually from questionnaires or checklists. Within PROFES, various checklists proved to be particularly useful for the continuous assessment trials.

V. Collect data and analyse results

The data for continuous assessment indicators should be collected during project execution as part of the data collection routines agreed in the measurement plan. A spreadsheet program such as Microsoft Excel may be sufficient for data consolidation and analysis. In PROFES project we have used the MetriFlame tool ([7]) for managing the measurement data and producing graphs for analysis sessions. MetriFlame also supports continuous assessment by providing links between GQM–based metrics and ISO15504 processes. The frequency of assessment data analysis varies but typically milestones in the project and GQM feedback sessions (cf. Section 0) are good candidates for a snapshot of process capabililty. Note that for some indicators there may be measurement data, but for some indicators a quick check on the process by a competent assessor is needed, as it is not cost-efficient to automate everything.

5.2 Example: Techniques for linking actual process indicators with BOOTSTRAP processes

This example illustrates the process of defining metrics for continuous assessment. Note that the final, detailed indicators are not in this example.

In general, a good place to start is to look for the work products related to the process and base practices. In this sense the ISO15504 Software Construction process – ENG.1.4 (cf. [5]) should be one of the easiest to trace (see Table 1). Three out of four base practices ask for existence of work products, i.e. files that are usually made during development. If these are in a Configuration Management system (here PVCS for example), then it should not be difficult to recall this information. Alternatively, just a list of planned work products or artefacts would be the basis for checking whether something is done or not. This could be in Lotus Notes, MS-Project, MS-Excel, etc.

Table 1: Example sources for continuous assessment indicators for the ISO15504 Software Construction process (ENG.1.4).

	Base Practices	N	P	L	F		Possible way of measurement	Example metrics
BP.1	Develop software units			x	X		From CM (PVCS) or controlled list	# of source files
BP.2	Develop unit verification procedures			x	X		From CM (PVCS) or controlled list	# of test cases/unit
BP.3	Verify the software units			x	X		From CM (PVCS) or controlled list	# of completed test cases/unit, # of review reports
BP.4	Establish traceability		x		X		A bit problematic – see below	

x=current value X=expected target value N=Not , P=Partially, L=Largely, F=Fully achieved

For establishing traceability (BP.4) this is usually not so straightforward. If the development is done within a comprehensive CASE environment or a requirement management is used, then traceability should not be a big problem. For a more ”normal” situation a suggestion would be as follows: Have a traceability mapping (document) related to each software unit where the relationship between the code, its design and relevant requirements are clear. Later, information about related testing materials should be added to this traceability mapping.

The base practice rating could then be based on the level of achievement although this should be done carefully (if at all). For example, if all planned software units are developed, the rating for BP.1 is F, i.e. base practice is fully achieved. In practice, the criterion for an F can also be less than 100%. However, one should be cautious for automatic rating because from existence alone it can not be decided whether some existing work products contribute to fulfilling the purpose of the process. For example, lets assume everybody does coding their own way. This will most likely create problems later, so even if all software units are done, one could question whether base practice BP.1 is completely fulfilled. So it depends on the situation. Therefore, it is essential that a competent assessor is making the judgement for process capability.

Now, let´s look at level 2 (see Table 2). For the Performance management attribute (2.1) it looks like most of the data could be gathered from the project plan, project report(s), and a defect management tool (here PSBugs).

Table2: Examples of continuous assessment indicators at capability level 2.

2	***Managed***	N	P	L	F	Possible ways of measurement	Example metrics
2.1	**Performance management attribute**						
2.1.1	Identify resource requirements			x	X	Estimate needed resources as part of project plan (MS-Project)	% of estimates done, checklist (estimates y/n)
2.1.2	Plan the performance of the process			x	X	Decide which data to use from project plan (MS-project)	Depth of a WBS, checklist (planning uptodate y/n)
2.1.3	Implement the defined activities			x	X	Decide which data to use from project plan/reports (MS-Project), PSBugs	# of source files vs. plan, effort vs. plan
2.1.4	Manage the execution of the activities		x	X		Review documents (Lotus Notes), PSBugs	# of replans, # of review docs
2.2	**Work product management attribute**						
2.2.1	Identify req. for the integrity and quality		x		X	CM (PVCS), Quality plan (in project plan)	# of requirements for integrity and quality
2.2.2	Identify the activities needed			x	X	Project plan (MS-Project)	# of quality activities, e.g. reviews
2.2.3	Manage the configuration of work products		x		X	CM (PVCS)	# of files under CM, existence of CM audit report.
2.2.4	Manage the quality of work products		x	X		Review materials (Lotus Notes), PSBugs	# of review docs vs.plan, # of corrective actions open

x=current value   X=expected target value   N=Not , P=Partially, L=Largely, F=Fully achieved

Most information for the performance management can be found using MS-Project data. For the Work product management attribute (2.2) the sources for data are mostly Configuration management system (PVCS), and tracking and review materials contained in MS-Project and Lotus Notes. Quality management information might also be contained as part of the project plan. Defect data (PSBugs) reveals how defects are uncovered and corrected giving information for determining the capability of software construction. For example, the unit verification procedures would probably need to be improved if there are repeatedly coding defects that are found in integration phase or later. Note that not all measurements need advanced tooling. For example, a project managers tick mark on “planning up to date” box on a checklist gives already processed information on management practice 2.1.2–Plan the performance of the process. Finally, it should be remembered that the data is only collected using this structure; a competent assessor is needed to make the process capability rating. For example, it depends on the assessors judgement of the situation what does it mean for management practice 2.2.4 to have a value of 75% on “# of review docs/plan”, and a value of 44 on “# of corrective actions open” (see Table).

5.3 Experiences

The preliminary experiences from trialing the continuous assessment approach within PROFES are encouraging. It seems to be feasible and useful to define and use measurements for assessment purposes. Early and interactive feedback of process improvement activities appears to provide a healthy monitoring atmosphere within the development projects. However, there are limitations of the approach, which can also viewed as prerequisites for continuous assessment. Firstly, there should be experiences of goal oriented measurement, and preferably an existing measurement programme where continuous assessment could be added. The process capability viewpoint can be useful in feedback sessions but setting up measurements only for continuous assessment does not seem to be a cost effective solution. Secondly, it is not easy to find measurements for continuous assessment from scratch. Tool support and guidance is needed to map the ISO15504 structure and indicators to actual, relevant measurements. There should be, for example, a list of possible metrics associated with each indicator. Finally, as it looks like that the scope of continuous assessment is largely defined by current measurement programmes, it seems to be difficult to extend the approach to cover processes in an organisation more widely. On the other hand, promising experiences were gathered from using various checklists for the data gathering purposes. Further research is needed to explore the use of checklists for continuous assessment.

6. Summary and Future Work

The paper has provided an overview of the continuous assessment concepts and related exploratory study in the PROFES project. The motivation has been to reduce assessment costs and use assessment more closely within product focused process improvement to provide better process visibility and better transparency to changes in product quality. The suggested approach integrates assessment and goal-oriented measurement using ISO15504/BOOTSTRAP as a specialised measurement instrument while introducing the powerful feedback mechanism of GQM. Extensive tool support facilitates continuous assessment but is not always needed. The results of the exploratory study in PROFES are still very preliminary, but under specific circumstances the approach seems to provide additional benefits with reasonable costs.

The continuous assessment work within PROFES will be continued in the application projects to gather experiences of using the approach. Also, the aim is to gather an initial set of indicators suitable for continuous assessment for a limited set of processes. These could be used as a starting point for planning continuous assessment in other companies. Finally, the MetriFlame tool will be extended to provide more support for continuous assessment by mapping measurement programme data and definitions to ISO15504 processes.

7. Acknowledgements

The authors would like to thank the members of PROFES consortium and especially Dietmar Pfahl and Matias Vierimaa for their valuable input for this paper. This work has been supported by the CEC through the PROFES project N^o 23239.

References

1. Basili, Victor R., Caldiera, Gianluigi, and Rombach, H. Dieter. “Goal Question Metric Paradigm”. In John J. Marciniak, editor, Encyclopaedia of Software Engineering, Volume 1, John Wiley & Sons, 1994, pp. 528–532.

2. Basili, Victor R., Caldiera, Gianluigi, and Rombach, H. Dieter. “Experience Factory”. In John J. Marciniak, editor, Encyclopaedia of Software Engineering, Volume 1, John Wiley & Sons, 1994, pp. 469-476.

3. Bicego, Adriana, Khurana, Munish, and Kuvaja, Pasi. “BOOTSTRAP 3.0 – Software Process Assessment Methodology”. In Proceedings of the SQM ’98, 1998.

4. Campbell, M., Järvinen, J., Thomson, H., Vernon, J. “Methods and Tools for Software Process Assessment and Improvement”. In the proceedings of The 5th European Conference on Software Quality, Dublin, Ireland, September 16-19, 1996, pp. 11-21.

5. ISO/IEC TR 15504-2 : “Information Technology - *Software Process Assessment - Part 2: A Reference Model for Processes and Process Capability*”. Technical Report type 2, International Organisation for Standardisation (Ed.), Case Postale 56, CH-1211 Geneva, Switzerland, 1998.

6. Latum, Frank van, Solingen, Rini van, Oivo, Markku, Hoisl, Barbara, Rombach, Dieter, and Ruhe, Günther. “Adopting GQM-Based Measurement in an Industrial Environment”. IEEE Software, 15(1), January 1998, pp. 78–86.

7. Parviainen, Päivi, Järvinen, Janne, and Sandelin, Toni, “Practical Experiences of Tool Support in a GQM-based Measurement programme”. In Software Quality Journal, Volume 6, No. 4, December 1997, pp. 238 - 294.

8. Rementeria, S., et al., “1995/1996 Software Excellence Survey: Model and Detailed Results Analysis”, European Software Institute, Technical Report ESI-1996-PIA/96282, 1996.

9. Solingen, R. van, Berghout, E., Kooiman, E., “Assessing feedback of measurement data: Practices at Schlumberger RPS with reflection to theory”. In proceedings of the 4th International Software Metrics Symposium, IEEE-CS, November 1997.

10. Solingen, R. van, Berghout, E., “The Goal/Question/Metric method: a practical handguide for quality improvement of software development”. McGraw-Hill Publishers, 1999.

11. SPICE Project Trials Team. “Phase 2 Trials Interim Report”. June 17, 1998.

SPECIFIC REQUIREMENTS FOR ASSESSING EMBEDDED PRODUCT DEVELOPMENT

Pasi Kuvaja[1], Jari Maansaari[1], Veikko Seppänen[2], Jorma Taramaa[2]
(in alphabetical order)

[1]Department of Information Processing Science, University of Oulu
P.O. Box 3000, Linnanmaa, FIN-90401 Oulu, Finland
[Jari.Maansaari@oulu.fi, Pasi.Kuvaja@oulu.fi], fax: +358 8 5531890

[2]VTT Electronics
P.O. Box 1100, Linnanmaa, FIN-90571 Oulu, Finland
[Jorma.Taramaa@vtt.fi, Veikko.Seppanen@vtt.fi], fax: +358 8 5512320

Abstract

In this paper, new requirements to enhance any ISO 15504 conformant assessment methodology for embedded systems development process assessment are presented. The findings of the paper were discovered in an ESPRIT Project called PROFES (PROduct Focused improvement of Embedded Software processes), where the specific characteristics of embedded systems development were analysed in three industrial organisations. The results of the paper are requirements to include product management, product life-cycle and some specific support processes in the assessment of embedded software producing units. These processes, which are defined and outlined in the paper, enhance regular assessment models by making them more applicable for assessing embedded product development organisations and projects.

1. Introduction

The number of products based on embedded computer systems has rapidly increased. At the same time, complexity of product features controlled or supported by embedded computers has dramatically increased and the role of embedded software has become crucial. All these developments make it necessary both to enhance existing processes and to define new ones in industrial embedded systems development. It is clear that embedded systems development includes specific features that cannot be found in traditional software development. It is thereby obvious that these features set new requirements for the process assessment and improvement methodologies, which have this far been developed mainly for non-embedded software applications in mind.

All assessment methodologies have a common goal to evaluate software processes against the best practices and produce a time-slice of the processes at that moment. All software assessment methodologies are general in their nature, and thereby do not address features that are specific to some specific domain. Additionally, most assessment approaches assume explicitly[4] that software development is either in-house or contract based development, which restricts their usability in the assessment of product development. This applies for example to all the CMM based approaches [SEI93, SEI95] that use the U.S Department of Defence standard (DoD8127A) [DoD88] for process definition.[5]

Limitations of current assessment methodologies to adequately support assessment and improvement of embedded systems development were recognised by industrial companies aiming at improving their embedded products. This led to the establishment of the PROFES project[6] [PRO97], whose main goal is to develop, validate and exploit a methodology for product quality driven software process improvement. The project combines and enhances well-known and widely used process assessment [KSK94], improvement [BC95] and goal-oriented measurement [BCR94] methodologies to form a new improvement methodology that identifies also product and process dependencies. In the paper findings of PROFES project are used in defining a set of new processes that are central in developing product based embedded systems. Scope of the paper is

---

[4] Although it has not been stated in any form, but only those processes are involved in the reference models of the assessment approaches.

[5] The standard focuses on the contract management and organisation's internal quality assurance processes.

[6] PROFES (PROduct Focused improvement of Embedded Software processes) is an ESPRIT Project (No 23239) lasting 1.1.1997 to 30.06.1999. The methodology providers in PROFES consortium are Etnoteam S.P.A from Italy, Fraunhofer IESE from Germany, University of Oulu, and VTT Electronics from Finland, and the application providers are LM Ericsson Finland, Dräger Medical Technology from the Netherlands, and Tokheim Retail Petroleum Systems from France.

focussed onto those processes directly related to product development, product management and support required in embedded product development. The results presented in this paper were derived through a literature analysis ([UlE95, Cas96, GeK96, HuD96, MWM97, SEP90, Sol95, SKO96, Sta96, SEI95, Boo97, JKK97]) and three industrial application experiments.

The composition of the paper is as follows. Section 2 defines basic concepts related to product development and section 3 introduces characteristics of embedded systems. Section 4 describes a defined general life-cycle model for developing embedded products and identifies key activities for each phase of life-cycle. Section 5 summarises findings of the paper and presents directions for future work.

2. Development of industrial products

The success of industrial companies in embedded systems business depends on their ability to identify customer needs and to create quickly products that meet the needs and can be produced at low cost. Achieving these goals is not only a marketing problem, nor is it a pure design problem or a manufacturing problem; but it is a product development problem involving all the functions. An industrial *product is* an artefact sold by an enterprise to its customers. *Product development* is a set of activities beginning with the perception of a market opportunity and ending in the production, sale, and delivery of a product. A *product development process* is a way to see product development as a product life-cycle conformed by a sequence of steps or activities that an enterprise employs to conceive, design, and commercialise a product. A general product development process is illustrated in Figure 1.

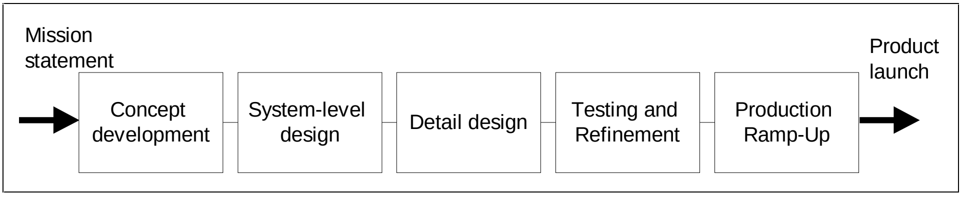


Figure 1. A general product development life-cycle [UlE95]

The product development life-cycle presented in the Figure 1, includes five phases: product concept development, system-level design, detail design, testing and refinement and production ramp-up. In the product concept development phase the needs of the target market are identified, alternative product concepts are generated and evaluated, and a single product concept is selected for further development. The system-level design phase includes the definition of the product architecture and the division of the product into subsystems and components. The product specification is completed in the detail product design phase by defining the geometry, materials, and tolerances of all of the unique parts of the product. The testing and refinement phase involves the construction and evaluation of multiple pre-production versions of the product. In the production ramp-up phase the product is manufactured using the intended production system. The purpose of the ramp-up is to train the work-force and to solve any remaining problems in the production processes.

The general development life-cycle described above will differ in accordance with a organisation's unique context. The general approach is most convenient in market-pull situation, where organisation begins product development with a market opportunity and then seeks out whatever technology is required to satisfy the market need. Technology-push, platform dependent, process-intensive and customised are the four other product development approaches defined by [UlE95]. Many products based on embedded systems are nowadays developed using the "platform dependent" approach, although the versatility of embedded systems product applications makes it hard to evaluate the exact portion of this approach. A platform product is built around a pre-existing technological subsystem, the platform, for which previous development investments have been made. The platform may consist of one or several distinct layers or smaller subsystems and modules. Therefore, every attempt is made to incorporate the platform into several different products. In this approach concept development, system-level design and detail design phases are affected by the reuse of the platform, and a process to maintain the platform must be established.

Expected application risks and quality characteristics of the product to be developed have also significant influence to product development process. It is quite clear that for example reliability and functionality requirements are different between different products. This fact has also strong effects to the quality characteristics of software to be developed. In a survey [SKO96] that was carried out among embedded systems professionals' world wide, reliability was indicated as the most critical requirement for embedded software. Maintainability and expandability were also seen quite important by many of the respondents of the survey. In the PROFES project the results of an inquiry among the experts of the application providers resulted that the three most important quality characteristics are reliability, maintainability, and cost-effectiveness. One of the key visions of the PROFES project is that *product quality improvement* in the domain *of embedded systems* will be achieved through the application of a methodology (a framework of methods and tools) for embedded systems software improvement that is driven by customer oriented product improvement needs.

3. Embedded computer systems

3.1 Embedded systems domain

Embedded systems are electromechanical systems relating electronics, mechanics, computer hardware and software closely to each other. They consist of control, communication and other intelligent functions implemented by using one or more computing modules. Embedded systems are usually built using several different technologies. In the subsequent figure the conceptual structure of embedded systems is outlined. Electronic products in which embedded systems are incorporated are the *target environment* of the systems. The primary task of the system is to control the target environment, very often within strict timing, resource consumption and reliability constraints. The control of the target environment is typically implemented using sensors and actuators that may themselves be embedded systems or unintelligent input-output devices. Unintelligence of the target environment is still typical in many embedded systems applications, although embedded control is rapidly diffusing to previously unintelligent appliances, such as push buttons, light bulbs, tickets and price tags to name just a few. The target environment is often *distributed*, i.e. there is a combined embedded control and communication problem to solve. Embedded communication is increasingly based on digital signal processing (DSP) technologies, and wireless communication technologies are rapidly emerging. The next generation of embedded systems will support *ubiquitous* *computing*, computers will be "everywhere" and communicate wirelessly with each others.

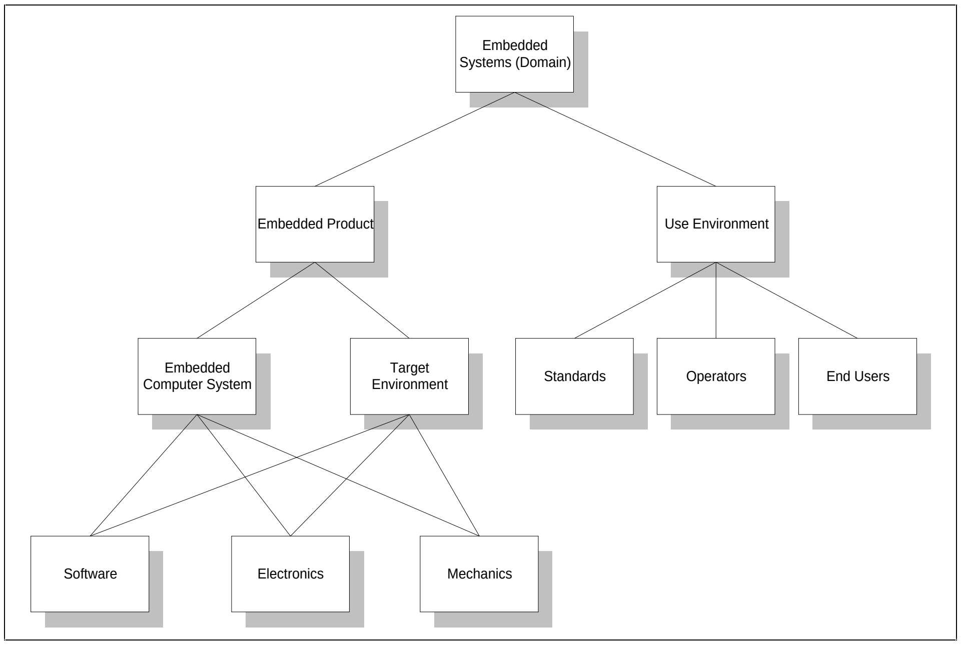


Figure 2. The conceptual structure of embedded systems.

The core technological parts of embedded systems are *embedded software* and computer hardware (Figure 2). Embedded software consists typically of a combination of built-in and general-purpose software used to control the target environment. Electromechanical products, where the role of mechanics is essential, are called *mechatronics applications*. Mechatronics is viewed as encompassing topics ranging from embedded control of automated devices and equipment to robotics and manufacturing automation. Embedded systems used in mechatronic applications are associated particularly with the enhancement of products, machinery and processes. Considering the three industrial case organisations of the PROFES project, automated gasoline pumps used at service stations are good examples of mechatronic applications, where reliability requirements of the embedded control are extremely high.

The use environment of products controlled by embedded systems includes end-users, operators and standards. *End users* range from non-technical to highly technical people who use the product for highly specific purposes, such as research scientists and medical experts. *Operators* are organisations or groups of people who organise, manage, support and make use of the product. A good example of an operator is a telecom operator who has established and manages and controls a telecom network for mobile telephones. *Standards* set a basis and regulate the use of many products in their use environments. A good example is a mobile communication standard, such as GSM that must be followed by all product developers.

Either the use environment or some standard or both may define the information content, or services to be provided by the product, and thereby affect directly to the characteristics of the embedded system incorporated in the product. Medical devices used in intense care units or operating theatres are examples of products, whose embedded control systems must be developed by taking into account both specific use environments and national and international standards and regulations.

3.2 Domain-specific characteristics of embedded systems

The modern embedded systems and their development very often include the following specific characteristics:

- software is closely connected to hardware;
- product design is constrained by the implementation technology;
- different technologies used in the same embedded system are developed in parallel;
- different low-level and high-level software implementation technologies are used;
- real-time processing and data management are used, and are often based on a distributed system architecture;
- reliability requirements are crucial;
- device autonomy is used;
- maintainability and extendibility through new functions, technologies and interfaces is often required;
- cost of the product is important in mass-markets;
- short development lead-time (time-to-market) is required.

The simultaneous development of a multitechnological implementation is typical to many embedded systems. The use of several different hardware, electrical and mechanical implementation technologies and their tight connection to the software is one of the central system characteristics. This creates needs to define which product functions will be implemented using which technologies. Although functions would have been successfully allocated to technologies, their further development requires understanding of both logical interconnections between functions and physical interfaces between different technological parts.

Most embedded systems involve tightly coupled hardware and software parts. Product markets and use environments set requirements which force their developers to tailor these parts to different purposes. Since relationships between software and hardware intensive parts are direct and inflexible, a change in one of the technologies may demand changes in the other technology. This means that changes often go through the interfaces of implementation technologies. As implementation technologies evolve rapidly, it is quite common to make new versions of the products by adopting new technologies. In both cases maintainability and extendibility of software become critical because the software have already been delivered to customers. Since reliability is essential in many applications and cost effectiveness is a driving force in electronic mass products, most technological and functional changes need to be addressed at the product level.

The development environments of embedded software have been quite specific and restricted, when compared to other software-intensive applications, which are executed on fully commercial computing platforms and developed using advanced design and testing tools. The main reasons behind are as follows: low-level programming languages, i.e. assembly languages have been used especially in time-critical parts of the software; only restricted ready-made system software solutions have been available, e.g. minimal or missing operating system functions; and testing environments must have been tailored for specific product deliveries.

Although the situation is improving in embedded systems development, many of the limitations are still valid. The use of more powerful microprocessors in high-end embedded systems has already made possible to use more advanced software development environments e.g. in aerospace, telecommunications and automotive industries. This has, however, created new needs to integrate data processing and application development environments to embedded systems and software design tools. High-end embedded systems include, on one hand, subsystems that come close to traditional data processing applications, such as workstation-based real-time server machines. On the other hand, they may also include such deeply embedded subsystems as intelligent sensors and actuators.

Embedded systems are generally real-time systems, which respond to events within specified time limits when controlling the target environment. Therefore they are quite different from traditional data processing applications where the computer system and the use environment may have a rather loose connection. In addition to this, many products that are controlled by embedded systems are required to have a good service level e.g. they have to be in use for twenty-four hours every day. Therefore, their functionality is characterised by high autonomy. This requires software solutions that are highly dependent on different phenomena and situations that may occur in the target and use environments.

The volume of embedded systems has rapidly increased in several application areas, e.g. in consumer electronics, telecommunication, and automobile industry. This has created high competition. The role of embedded systems has often become essential in the determination of the product cost. However, the increased cost of using more efficient embedded systems has to be balanced by the willingness of the customers to pay for the increased functionality that the new product generations provide.

4. Embedded product development processes

The processes that are required in the assessment of embedded systems development are those needed to manage the relationships between the company, markets and mass of customers. The following three process classes were identified through a literature analysis and three application experiments:

- product management processes,
- product life-cycle processes, and
- supportive processes.

4.1 Product management processes

The main goal of the product management processes is to guarantee a competitive portfolio of products that creates and satisfies customer and market needs. The more specific goals are *customer satisfaction* and *product profitability*. Product management processes assess product goals, strategies and plans, manage the decision process for changes in the product portfolio, and provide product information to the other processes of the producing unit[7]. Product management processes contribute mostly to *time to market* and *customer lead time* quality characteristics. They are also expected to contribute in improving the market trust, reaching high level of customer satisfaction, and increasing the profitability of the product. Potential internal effects of the product management processes onto the product producing unit are improved efficiency, increased individual motivation, and improved quality of work. The product that the product management processes are managing might be recognised at least from the following viewpoints: customer solutions, engineering requirements, product information, pricing, distribution, and service. The product management process class *contains product strategy formulation,* and *reuse* processes.

The purpose of the *product strategy formulation* process is to continuously assess the market needs and specify properties and product business plans for each product area. Successful execution of the process defines the marketing strategy, evaluates the implementation technologies available and plans the product options, identifies the suppliers, defines the product assembly scheme, evaluate the economic feasibility of the production, and identifies the product maintenance strategy.

The purpose of the *reuse* process is to promote and facilitate the reuse of new and existing software work products from an organisational and product/project perspective. The activities of the reuse process are to define reuse strategies, to identify and establish reuse activities, to identify reusable entities, and to establish reuse infrastructure.

---

[7] The unit within an organisation where the product is produced and/or maintained, often acronymed PPU.

4.2 Product life-cycle processes

The product life-cycle processes are: *product requirements specification, product design, systems design and implementation, systems integration and testing, and production and installation*. The following figure (Figure 3) illustrates the life-cycle processes that form a logical order of execution.

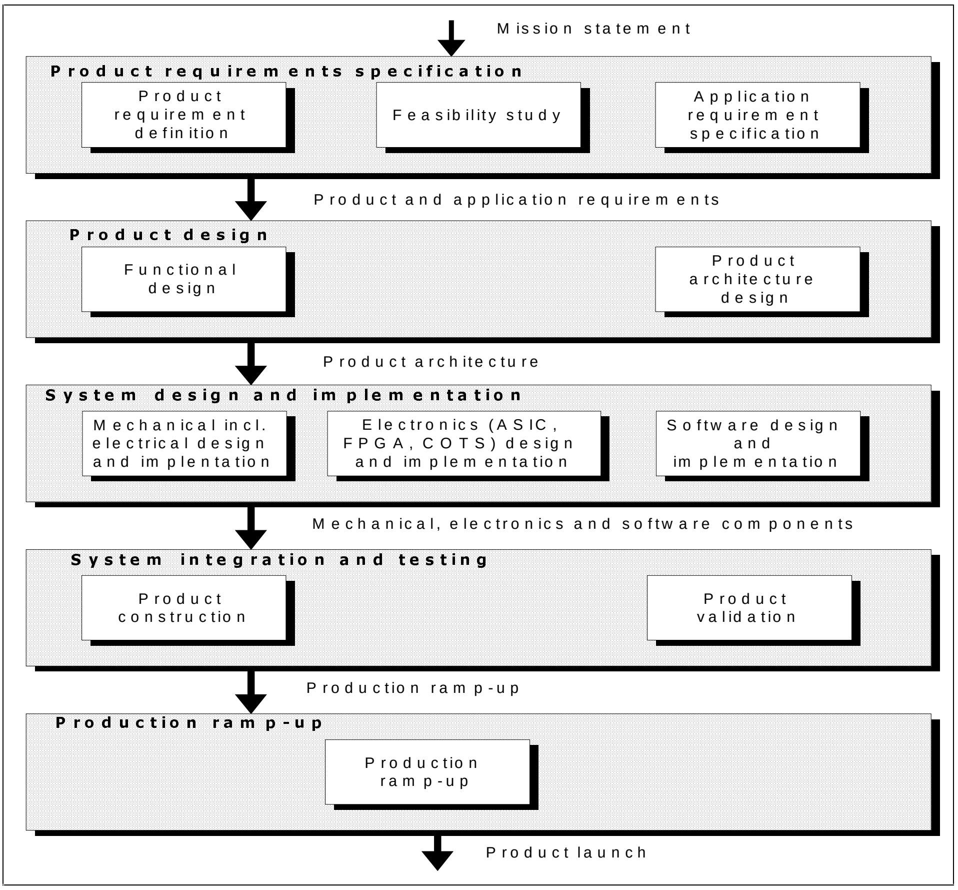

Mission statement
Product requirements specification
Product requirement definition
Feasibility study
Application requirement specification
Product and application requirements
Product design
Functional design
Product architecture design
Product architecture
System design and implementation
Mechanical incl. electrical design and implentation
Electronics (ASIC, FPGA, COTS) design and implementation
Software design and implementation
Mechanical, electronics and software components
System integration and testing
Product construction
Product validation
Production ramp-up
Production ramp-up
Production ramp-up
Product launch

Figure 3. Life-cycle processes for embedded product development

The purpose of the product requirements specification process is to define the product requirements, and ensure both economical and technical feasibility of the product to be developed. The product requirements are defined in co-operation with people from marketing, manufacturing, service & support and R&D, and then documented in the *product requirements specification*. People responsible for collecting information and feedback from the customers play an important role in this process. Product requirements specification process includes three main activities that are: *product requirements definition, feasibility study*, and *application requirements specification.*

Main objective of *product requirements definition* is to identify possible business opportunities and develop/evaluate preliminary product ideas according to these opportunities. The product management team is responsible for collecting product requirements from the customer. Different types of requirements (i.e. electronics and software requirements) may already be separated by the product team and passed to the project management team. Besides this, for those ideas chosen for further development a business plan is developed in this phase. *Feasibility study* involves activities related to analysing initial product requirements and product ideas, evaluating available technologies and designs, risks and costs. Product specification document and project management plan is then developed. Product specification document contains information related to product architecture, project plan, risk analysis, and project budget. In *Application requirements specification* requirements for product to be used in a specific market are defined. These requirements are also collected from customer as product requirements but they are specified to special market

Main objective in *product design* process is to define functional characteristics and architecture of the product. The role of R&D and manufacturing is emphasised in the phase where the system level architecture of the product implementation plan is defined with specifications of the software, electronics, mechanics, etc. Product design process includes two main activities that are *functional design,* and *product architecture design*. In *functional design* the functionality of the product is defined in detail. This is done by analysing functions of the system in terms of behaviour, structure and cost. User manual is also initiated in this activity. In *product architecture design* software, electronics and mechanics designs are refined, verified and validated. User manual is updated and the release of the product is decided for further development.

All components (including software, electronics and mechanical) of the product are designed and implemented in *systems design and implementation* process. From the viewpoint of the electronics development, the process has been well established since the 1970's and has been producing quality products up to date. As the role of software has evolved, its size has increased remarkably, often taking majority of resources allocated for product development work. This has increased the needs for product and project management in terms of understanding the dynamics of software development more comprehensively.

Today the hardware development process includes circuit design, printed circuit board (PCB) layout design (using specific computer-aided electronics design automation (EDA)), and hardware and electronics production. ASIC development has become common. It resembles software development, since the design phase is based on the application of specific hardware design languages (VHDL). The development of mechanics includes analysis and design of dynamics, motion, and control problems. Computer-aided design tools provide, for example, design optimisation and simulation support. Behaviour or algorithm design is a specific task in the development of embedded systems. In mechatronic applications algorithms are used to control the target environment which can include, e.g., hydraulics, pneumatics, and electronics. In telecommunication, algorithms support e.g. digital signal processing. In electronic instruments they are used, for example, to analyse measurement data.

In the *systems integration and testing* process, the product components are integrated with several testing steps before the whole product is tested. The last testing phase may include extensive field-testing and certification before the product is ready for production and installation. Due to the close connection of several technologies, concurrent engineering principles have become an increasingly important means to manage the development of embedded systems. Systems integration and testing phase includes following main activities: *pilot product construction, and pilot product validation*. Pilot product is constructed by integrating mechanical, electronic and software components and by executing integration tests. The product documentation including user manual is completed in this activity. Pilot product is validated by executing function and field tests. Appropriate corrective actions are taken according to test results.

In the *production ramp-up* process the product is manufactured using the intended production system. The purpose of the ramp-up is train the work-force and to work out any remaining problems in the production. The products produced during production ramp up are in some cases supplied to preferred customers and are carefully evaluated to identify any remaining flaws before the actual production is started.

4.3 Supportive processes

Although the main focus of the three PROFES application experiments was on life-cycle processes, some processes related to support area impact all the processes. The supportive process class contains the following *processes: measurement*, *process control*, and *product control and promotion*,

The purpose of the *measurement* process is to collect and analyse data relating to the products developed and processes implemented within the organisational unit, to support effective management of the processes, and to objectively demonstrate the quality of the products. Successful execution of the process results the following outcomes: An appropriate set of measurements, driven by the project and organisational goals, will be identified. All data required will be collected and analysed. A collection of historical data relating to process implementation will be established and maintained. Measurements will also be used to support decision making and provide an objective basis for communication between the interested parties.

The purpose of the *process control* process is to collect and analyse data relating to the performance and management of the processes within the organisational unit, to support effective management of the processes, and to monitor the effects of process changes. When the process is successfully executed process data is collected and analysed, performance and management of the processes is evaluated, and process performance and changes are monitored.

The purpose of *product control and promotion* process is to provide other processes with relevant product information, such as release information and facts. When the process is successfully used commercial support and customer contacts will be set, promotion and training material is developed, product is designed, and production environment is prepared.

5. Summary

Findings of the PROFES project that are reported here, outline embedded systems specific enhancements to any ISO 15504 conformant assessment approaches[8]. The enhancements were identified in a literature analysis and inquire among the industrial partners of the project. In this way the processes that were discovered are those actively used and existing in the development of modern embedded systems. The enhancements were implemented in the BOOTSTRAP methodology and validated in the assessments performed in the three embedded applications of the industrial partners.

The new processes cover the product viewpoint that is not explicitly present even in the becoming standard for software process assessment and improvement (ISO 15504) or in any commercial assessment methodologies (for example in CMM). The new processes define a new process area of product life-cycle, product management processes and new supportive processes.

6. Future efforts

It is quite clear that more work is needed to validate and incorporate the new processes into embedded systems assessment and improvement. In fact, some validation work has already been done by applying the approach defined earlier in the SPICE project[9]. Another line of future research effort is needed to discover whether the new processes defined here will cover the increasing needs of the future embedded systems that are based on platform solutions. Already today a number of embedded systems based products are developed using platform solutions and the number is rapidly increasing. One way to continue might be to define new or enhance the current processes to cover the evolution and validation of the embedded system platform, and reuse of platform elements (like requirements, specifications, designs or ready-made implementations)

[8] In the PROFES project, BOOTSTRAP assessment methodology was used.

[9] To speed up the work to develop a material to become a basis for a new standard (ISO 15504), ISO/IEC/SC7set a working group (WG10) to carry out the development stage through an international project called SPICE (Software Process Improvement and Capability dEtermination). See also [PaK94].

Reuse contains screening, selection, modification and integration of platform elements. Several support processes not addressed in this paper, such as configuration management, will also become more complicated in platform-based embedded systems development.

7. Acknowledgements

The research, whose results are described in this paper, was carried out in the Esprit4 project PROFES (EP 23239) funded by the European Union. In addition to VTT Electronics, the University of Oulu and LM Ericsson Finland, Dräger Medical Electronics (the Netherlands), Etnoteam (Italy), Fraunhofer IESE (Germany) and Tokheim Retail Systems (the Netherlands) participate in the project. They have also contributed to the research reported in this paper. The financial support of Tekes for the Finnish parties of the PROFES project is gratefully acknowledged.

References

[BB94] Bache, R. and Bazzana, G. Software Metrics for Product Assessment, McGraw-Hill Book Company, London, UK, 1994, 248 pages.

[BCR94] Basili V.R., Caldiera G. and Rombach H.D. Goal Question Metric Paradigm. In J.J. Marciniak, ed., Encycl. of SW Eng., vol. 1, pp. 528–532. John Wiley & Sons, 1994.

[BCR94a] Basili V.R., Caldiera G. and Rombach H.D. Experience Factory. In J.J. Marciniak, ed., Encycl. of SW Eng., vol. 1, pp. 469–476. John Wiley & Sons, 1994.

[BC95] Basili V.R., Caldiera G. Improve Software Quality by Reusing Knowledge and Experience. Sloan Management Review, pp. 55-64, Fall, 1995.

[BiJ98] Andreas Birk, Janne Järvinen, Seija Komi-Sirviö, Markku Oivo, Dietmar Pfahl, PROFES – A Product-driven Process Improvement Methodology, In Proceedings of the Fourth European Software Process Improvement Conference (SPI ‘98), Monte Carlo, Monaco, December 1998.

[Boo97] Bootstrap Institute. Bootstrap v3.0: Technical Overview, 1997.

[Cas96] Castelli, G. Software Architectures for Deeply Embedded Systems: The OMI Approach, in Proceedings of An International Symposium On-board Real-time Software, ESTEC, Noordwijk, The Netherlands, November 1995, ESA, SP-375, pp. 87-93.

[ESA91] ESA Software Engineering Standards ESA PSS-05-0. Issue 2. ESA Board for Software Standardisation and Control, European Space Agency, Paris (February 1991).

[GeK96] George, G.W. and Kryal, E. The Perception and Use of Standards and Components in Embedded Software Development - A report for the OMI Software Architecture Forum, July 1996, Draft, 28 p, the www address: http://www.osaf.org/resource.html, July 1996, Draft, 28 p.

[HuD96] Hurst, W. and Dennis, J. OMI Software Architecture Forum (OSAF), Report on the major issues and concerns of industry group associations and their members on the future use of embedded microprocessors within their respective industries, - A report for the OMI Software Architecture Forum, July 1996, 53 p, the www address: http://www.osaf.org/resource.html.

[ISO89] ISO 9001. Quality Systems. Model for Quality Assurance in Design/Development, Production, Installation and Servicing. International Organisation for Standardisation, Geneva, 1989.

[ISO91] ISO/IEC. Information technology – Software product evaluation – Quality characteristics and guidelines for their use. International standard 9126, ISO/IEC Copyright Office, Geneva, Switzerland, 1991.

[ISO91a] ISO 9000-3. Quality management and quality assurance standards. International Standard. Part 3: Guidelines for the Application of ISO 9001 to the Development, Supply and Maintenance of Software,ISO, 1991.

[ISO95] ISO/IEC. ISO-12207, Software Life-cycles, Version 1.00, International standard 15504, ISO/IEC Copyright Office, Geneva, Switzerland, August 1995.

[ISO98] ISO/IEC JTC 1/SC 7, ISO/IEC TR 15504-2:1998(E), Information technology – Software process assessment – Part 2: A reference model for processes and process capability, Technical Report type 2, ISO 1998.

[JKK97] Järvinen, Khurana, Kuvaja, and Saukkonen. The Role Of Embedded Software In Product Development, Conference on Software Quality Management (SQM'97), Bath, UK, September 1997.

[KSK94] Kuvaja, P., Similä, J., Krzanik, L., Bicego, A., Koch, G., and Saukkonen, S., Software Process Assessment and Improvement. The BOOTSTRAP Approach. Blackwell Business, Oxford, UK, and Cambridge, MA 1994.

[KuB93] Kuvaja, P., and Bicego, A., BOOTSTRAP: Europe's assessment method, IEEE Software, Vol. 10, Nr. 3 (May 1993), pp. 93-95.

[MWM97] Masera, M., Wilikens, M. and Morris, P. (eds.) Dependability of Extensively Deployed Products with Embedded IT, Full Report of the Workshop help on 21-22 November 1996, Brussels, Belgium.

[PaK94] Paulk, M.C., Konrad M. D., An overview of ISO's SPICE project, American programmer, February, 1994, pp. 16 - 20.

[PRO97] The PROFES project, URL: http://www.ele.vtt.fi/profes.

[SEI93] Paulk, M., et al. Capability Maturity Model for Software, Version 1.1, CMU/SEI-93-TR-24, Feb. 1993.

[SEI95] Software Engineering Institute, A Systems Engineering Capability Maturity Model, Version 1.1, SECMM-95-01, CMU/SEI-95-MM-003, November 1995.

[SEP90] Seppänen, V. Acquisition and reuse of knowledge to design embedded software, VTT Publications 66, Espoo, Finland 1990.

[SKO96] Seppänen, V., Kähkönen, A-M., Oivo, M., Perunka, H., Isomursu, P. and Pulli, P. Strategic Needs and Future Trends of Embedded Software, Technology review 48/96, TEKES Finland, October 1996.

[Sol95] van Solingen, R. and van Uijtregt, S. Partnership with Customers in Product Improvement - Testing Embedded Software Products in the Field, 3rd International Conference on Reliability, Quality & Safety of Software-Intensive Systems (ENCRESS'97), Athens, Greece, May 1997, D.Gritzalis (ed.), Chapman & Hall, pp. 201-214.

[Sta96] Stankovic, J.A. Real-Time and Embedded Systems, ACM Workshop on Strategic Directions in Computing Research, ed. by Peter Wegner and Jon Doyle, MIT laboratory for Computing Sciences, Cambridge, Massachusetts, June 1996,

[UlE95] Ulrich K.T. and Eppinger S.D., Product design and development, McGraw-Hill, Inc., Singapore 1995.

Product focused SPI in the embedded systems industry

- Experiences of Dräger, Ericsson and Tokheim -

Rini van Solingen[10], Tokheim and Eindhoven University of Technology, The Netherlands

Pieter Derks, Dräger Medical Technology, Best, The Netherlands

Jorma Hirvensalo, Oy LM Ericsson Ab, Jorvas, Finland

Abstract

Software specific problems have been handled in the software community through focusing on the software process, and continuous improvement of that software process. However, the contribution of software process improvement (SPI) to product quality has not been proven yet. The PROFES project customised successful approaches into one embedded systems specific SPI methodology focused on improving product quality. This PROFES improvement has been fully applied in three embedded systems developing organisations: Dräger Medical Technology, Ericsson and Tokheim, in three industries (telecommunication, medical systems, and petroleum retailing).

The main message from these applications is that PROFES really helps in focusing to product areas that have a priority for improvement. The companies strongly support that only effort is spent on product attributes that are relevant. Quality of the final product is the central objective, which is highly appreciated, since the embedded system itself is being sold and not the development process that created it.

[10] The authors can be contacted via: R.v.Solingen@tm.tue.nl

1. Introduction

Software problems have been handled in the software community through focusing on the software process, and continuous improvement of that software process. However, the contribution of software process improvement (SPI) [6] to product quality is still an assumption and has not been proven yet.

This paper presents the results of applying the PROFES methodology in three industrial organisations: Dräger, Ericsson and Tokheim. These companies have been involved in the EU-Esprit project PROFES (23239), which customised successful approaches [1][3][8][9] into one embedded systems specific methodology that links product quality objectives directly to the software development process. Note that this paper presents the results and experiences of PROFES application and not the process of PROFES application, because the methodology is already published in other publications [3], and the individual company process will be published separately (see for example the Tokheim process in [9]).

Reason to start developing the PROFES methodology was the notion that current software process improvement standards, such as the CMM, Bootstrap, Spice, or GQM insufficiently address the product quality objectives, when applied in the embedded software industry. As the embedded software industry sells products and not processes, a more product centred approach should be used in this domain when applying software process improvement. However, such a product focused SPI methodology was not available, which was the reason to start PROFES. On the other hand, many successful, ‘proven’ approaches were available, each focusing on one specific aspect of product focused SPI. Therefor it has been decided to develop the PROFES methodology not completely from scratch, but integrate existing approaches. In that sense, PROFES supports also companies that already have SPI activities such as assessments, improvement plans, process models or measurement in place.

The PROFES methodology is intended to be customisable. This is necessary because each company or project has its own objectives. A ‘one-size-fits-all’ method to product focused SPI is expected not to fulfil these individual needs. It is for example possible that a department has an objective such as ‘reaching level 2 at the end of next year’, or an objective such as ‘the first version of the product may have only five defects’. Such different objectives should also be handled differently by an improvement approach.

The PROFES methodology contains several components such as assessments, measurement, experience factory, process modelling, etc. that are available as a kind of toolbox. From this set of components an organisation can select the mix that fits best. Different starting points, together with the (product) objective, trigger different ways to carry out an improvement programme. PROFES methodology supports these differences very well.

The main parts of the improvement programmes in the industrial companies are focused on the objectives set in the individual projects. Based on these objectives, improvement actions have been carried out which will be explained, together with the results that were observed and measured in the projects.

2. Dräger improvement objectives and results

Dräger is a 1.4 billion DM multinational operating primarily in the fields of medical technology and safety technology, with limited operations in aerospace technology. It has about 8100 employees, of which over 5300 are employed in Germany. The three divisions of Dräger are Medical Technology, Safety Technology and Aerospace. The core business of Dräger Medical Technology are the development, production and service of gas monitors, single and multi-parameter patient monitors, fluid pumps, incubators and defibrillators for application in anaesthesia, intensive care, neonatal and emergency care.

2.2 Dräger MT development project

Dräger MT-M (the monitoring sub division of Dräger MT) is developing a complete new line of patient monitoring devices. This family of devices should create a BSW (BedSide Workstation) around each bed location in a department in a hospital. The BSW’s are intended for the intensive care stations as well as the operation room area. The system incorporates network connections between various elements of the system, and allows patients data exchange and viewing a patients data at several locations. The development project is organised in a project. The development activities take place on two sites: in Lübeck, in Germany, and in Best in the Netherlands. The PROFES improvement methodology has been applied within Best.

2.3 Product improvement objectives

Based on many years in the medical equipment business and recent market explorations, the following improvement objectives where derived for the product.

- Higher **reliability** of the overall product. This means, a lower number of defects in the final product during operation by the end users.

- Higher **fitness** for use of the overall product. Meaning, the product should give more functions required by the end users and be able to better support their processes.

- Higher **predictability** of the quality, time and costs of the development of the product.

2.4 Improvements carried out

The potential process improvements within the Dräger organisation are indicated by process assessments, according to BOOTSTRAP [2]. The process improvements carried out are selected on their (expected) contribution to the quality of the final product.

2.4.1 Process improvements to improve product reliability

- Incremental development. To be able to get an early feedback on the product quality, the products are developed in so called increments. Each of these increments take about six months and result in a working prototype, featuring a subset of the final functionality. These prototypes are tested in hospitals, to get the required feedback.

- Testing. To verify the (high) quality requirements, an adequate test strategy is implemented. Also an independent test group is installed.

- Inspections. To improve the reliability of working products, Fagan inspections are applied on analysis documents, design documents and test specifications.

- System and Software architecture. An adequate system and software architecture is defined. Time and money is explicitly allocated to enable these activities.

2.4.2 Process improvements to improve product fitness for use

- Co-operation between development and product marketing. To ensure realistic products specifications, the specifications are made in close co-operation between the development and the product-marketing department.

- Buy in system modules. To be able to offer state of the art functionality, some system modules are bought in from world-wide-recognised market leaders on patient monitoring technologies.

2.4.3 Process improvements to improve development predictability

- Continuous integration. To prevent unpredictable outcomes of the developments, the various parts of the product are integrated and tested as early as possible.

- Sub contract management. Because of the shift to system integration, the quality, time and costs of the development largely depends on the various suppliers. To manage this adequately, sub contract management is selected as one of the focus areas.

2.5 Results of the improvements

The results of the process improvements can first of all be found in the processes themselves. As the market release of the products is planned for November 1999, results are not available yet. However, the quality of the available prototypes is a reasonable indication for the quality of the final products.

2.5.1 Processes

With respect to processes the results of the improvements are as follows:

- BOOTSTRAP level. The BOOTSTRAP level of the processes increased from 0.5 to 2.5 on departmental level and from 2 to 3 on project level.
- Awareness of quality. The quality awareness increased, both for processes and for products. Clearly visible is the focus of the engineers on (improvements of) processes that lead to improved product quality.
- Monitoring of defects. The capability to monitor defects improves significantly. This enables an effective and efficient fixing of defects.

2.5.2 Products

With respect to products the results of the improvements are as follows:

- Early feedback. Due to the incremental development approach, it is possible to get early feedback on the product quality from the end user point of view.
- Increment II finished in time. Mainly because of the continuous integration activities, the increment II development is finished on the planned data.
- Functionality prototype II close to final. The functionality at the end of increment II, proved to be close to final in the second hospital field tests.

- Only 4.75% defects in field tests. From all found defects, only 4.75% are found during a field test in a hospital. This is considered to be a good result.

3. Ericsson improvement objectives and results

Ericsson manufactures products in almost all sectors of the telecommunications field and its 100,000 employees are active in more than 130 countries. Ericsson has approximately 14,000 software engineers working in more than 50 design centres. The core business of Ericsson is to develop and integrate telecommunication equipment. The PROFES application has been executed in the Telecom R&D division of Ericsson Finland.

3.1 Ericsson development projects

PROFES application was carried out in two projects. *The MACRO project* contained several sub-projects developing software for the AXE switching system. MACRO implemented the ISUP (ISDN User Part, an International Telecommunications Union standard) functionality of the signalling network. *The ITAS project* further developed the charging functionality in order to make it possible for operators to handle Interoperator Tariff Account Settlement (i.e. how to share toll-ticketing incomes between companies).

3.2 Product improvement objectives

Product qualities for interest of improvement were:

- Product **reliability**.

- Product **maintainability**.

The primary quality goal from customer’s perspective is a high In-Service Performance (ISP) including quantitative targets to shorten system down time and line down time.

3.3 Improvements carried out

Improvements were carried out in those processes that contributed most to the product objectives. Several process changes were implemented, including:

- More attention was put on capacity/non-functional requirements.
- The results of requirement analysis/interpretation were revised based on measurements of requirement clarity.
- Overall quality assurance got more attention.
- Improved design-test co-operation (designers participate in test reviews, testers participate in design reviews).
- Described and measured inspection criteria was used.
- Progressive monitoring of project by introducing new metrics.

3.4 Results of the improvements

Design quality expressed as mean fault density measured in Function test shows a significant improvement in comparison with baseline and goals. Fault density outcome (in kilo non-commented source statements) is much lower than in the preceding project. The goals were met in both projects. One cause for lower fault density in function test might be that software products were carefully desk-checked. Detection efficiency in number of faults per desk check effort was in the ITAS project twice as high (0.60 faults/hour) as in the MACRO project. MACRO product has passed the first 6 months at the customer. Only 3 major faults on operational performance were detected. Fault analysis has also shown that only one fault detected by the customer was received.

Results are also available on maturity improvement. Two Bootstrap assessments were held, the first one took place in June 1997 and the second one in June 1998. The process, in which we were able to observe a significant improvement, was Verification (SUP.4) process (incl. inspections). Comparison of process attribute profiles between MACRO and ITAS showed a significant growth of capability from 1.8 to nearly level 3. This was mainly achieved due to well planned and tracked inspection activities that were supported by well-established procedure.

4. Tokheim improvement objectives and results

Tokheim is worldwide leader in providing systems and services for self-service petrol stations. Tokheim has a revenue of 750 million US$, and 4,800 employees. Products of Tokheim are Fuel Dispensers, Point of Sales, EFT equipment, Back-Office and Forecourt Controllers. The Tokheim site supported with the PROFES methodology is located in Bladel, The Netherlands.

4.1 Tokheim development project

The development project guided with PROFES methodology was the World Wide Calculator (WWC) project. This project develops the central control unit, which should fit and function for all dispenser types and ranges for the new product family of fuel dispensers. The calculator is the central measurement and control unit in a dispenser, which displays the amount of fuel and money of a fuelling transaction to the customer. Beside this display function, a calculator controls the whole dispenser, meaning it controls the pump motors and valves, measures the flow rate, and communicates with a station control system or outdoor payment terminal.

4.2 Product improvement objectives

The main objective of the WWC project was a product cost reduction. Beside this very strict cost target, there were also severe deadlines for the project. This in all made the project team aware that product quality was highly at risk. Therefor the product improvement objectives within PROFES were mainly focusing on product **reliability**.

4.3 Improvements carried out

The focus on product reliability made the team aware that the focus was on detecting and eliminating product defects. Therefor, the improvements focused on testing. Within the WWC project the following improvements/changes were carried out that contributed to the product reliability objective:

- Much time in the design phase. More effort was spent during the design phase of the project. This was necessary due to the combination of extreme product cost target, while product reliability should be high as well. Therefor a very thorough and robust design was needed to fulfil these targets. This resulted in a highly structured product architecture.

- High amount of software reuse. Application of available software from a previous (intermediate) product did not only support in faster progress of the project, but also in the use of already stable and reliable software in the new product.

- Time for testing. Additional time for testing was created to assure the product reliability targets. This testing effort was spent on unit testing as well as system testing

- Dedicated resource for testing. In addition to the more testing time and effort, one engineer was assigned only to product testing. Because of this dedicated test person, parallel development and testing was carried out and not postponed to the end.

- Cross-personal testing. Instead of development and testing of units by the same engineer, the project team introduced cross-personal testing, which means that a different engineer than the one whom developed it tests the software.

- Closer co-operation with QA department. Due to the risks on product quality, the WWC project team requested a more frequent and closer involvement of the QA department than usual. This resulted in weekly meetings between the QA manager and project manager, focusing on process and product quality topics.

- Personal commitment to product quality. All project team members clearly committed to product quality. This ‘personal quality’ was one of the main success factors.

4.4 Results of the improvements

The final product contained a very low number of defects during the first field tests with the product. Only 2 defects were detected during the field test. These defects could not have been found during the in-house testing process, since these defects were rather complicated. However, the in-house testing process has been changed as such that similar defects will be detected in the future.

The product cost reduction objective was reached. Although this was not one of the objectives considered within the PROFES targets of the project, this result does contribute to the success of the project. Also the product was delivered within the planning limits, and therefor did not block the overall project on the development of the new dispenser product family.

5. Experiences with PROFES application

PROFES methodology is highly usable. During application of the PROFES methodology it appeared to be an approach that is highly usable in practice. None of the companies had problems in applying the methodology.

Customisability of the methodology is major benefit. A strong point of PROFES is that it a customisable approach to process improvement. Dependent on the needs on objectives of a company the methodology can be applied. Some companies may have a strong focus on improving along a scale such as CMM or BOOTSTRAP. This is very well supported by PROFES. On the other hand there are companies that use a product centred approach, which is also very well supported by PROFES.

Product focused improvements are feasible and do pay off. Centring all process changes around the expected impact on the product is feasible. Especially in the embedded systems area this is a critical success factor. The link between process and product is constantly evaluated and analysed.

There is still a lot to learn on product process dependencies (PPDs). The way in which PPDs work in practice, how their effects can be optimised, which critical context factors are present, is still unknown. Effects of PPDs appear to differ over organisations: what works in one organisation does not necessarily have to work in another. Past project experiences are sound input for PPDs.

Organisation and projects motivated to apply PROFES elements. The project teams supported the application of PROFES. This was due to the components of the methodology, to the goal-oriented character, and product orientation.

Recommendations from assessments were useful. The proposed improvement changes from the assessments were clearly accepted, not only for the individual projects but also on the organisational level.

Feedback sessions were valuable. Measurements trigger discussions within project teams, and facilitate a group learning process. However, more time is needed for measurement data analysis. The full integration of measurement in PROFES methodology made available data already useful during the project.

6. Conclusions

Due to PROFES the embedded systems industry can now apply a product focus in their process improvement programmes. Embedded system producers sell products, not processes. These products not only consist of software, but also of hardware. Improvements in the process should therefor always be aimed at an improvement in the product area: cost, quality or timeliness.

PROFES methodology helps in focusing to those improvement areas that are relevant for the specific organisation or project. Only effort is spent on product attributes that have a priority for improvement. PROFES methodology showed to be a useful and powerful approach to apply product focused SPI in practice. Based on the specific needs of the individual company and the specific development project, the applied improvement approach was customised fully supported by the PROFES methodology. This is considered being the strongest benefit of the PROFES methodology.

Results in the companies were revolutionary. Dräger was able to develop their product exactly within a very tight schedule, and this product was very positive received by hospital staff during the clinical tests. Dräger also increased its development project maturity from almost level 2 to level 3 in less than one year. Ericsson delivered their product with a design quality higher than their baseline. Tokheim supported a reliability critical project with a product reliability focused process improvement programme, resulting in just 2 defects.

7. Acknowledgements

The authors would like to thank Erik Rodenbach for his comments to an earlier version of this paper. Furthermore, thanks to all members of the PROFES consortium for their input and co-operation in the project. The work presented in this paper is sponsored by the European Commission through the PROFES project (23239).

References

[1] Basili, V., Caldiera, G., and Rombach, D., “Experience Factory” & “Goal Question Metric Paradigm”, In John J. Marciniak, editor, Encyclopaedia of Software Engineering, Volume 1, pages 469–476 & pages 528–532. Wiley & Sons, 1994.

[2] Bicego, A., Khurana, M., Kuvaja, P., BOOTSTRAP 3.0: Software Process Assessment Methodology, Proceedings of the SQM ’98, 1998.

[3] Birk, A., Järvinen, J., Komi-Sirviö, S., Oivo, M., Pfahl, D., PROFES: A Product-driven Process Improvement Methodology, In Proceedings of the Fourth European Software Process Improvement Conference (SPI ‘98), Monte Carlo, 1998.

[4] Hamann, D., Järvinen, J., Birk, A., Pfahl, D., “A Product-Process Dependency Definition Method”. In Proceedings of the 24th EUROMICRO Workshop on Software Process and Product Improvement. Pages 898-904. IEEE CS, Sweden, August 1998.

[5] Hamann, D., Järvinen, J., Oivo, M., Pfahl, D., “Experience with explicit modelling of relationships between process and product quality”. Proceedings of the 4th European Software Process Improvement Conference, Monte Carlo, 1998.

[6] Humphrey, W., Managing the Software Process. Addison Wesley, 1989.

[7] Latum, F. van, Solingen, R. van, Oivo, M., Hoisl, B., Rombach, D., Ruhe, G., Adopting GQM-Based measurement in an Industrial Environment. IEEE Software, 15(1):78–86, January 1998.

[8] Solingen, R. van, Berghout, E., The Goal/Question/Metric Method: A practical guide for quality improvement of software development, McGraw-Hill, 1999.

[9] Solingen, R. van, Uijtregt, A. van, Kusters, R., Trienekens, J., ‘Tailoring product focused SPI: Application and customisation of PROFES in Tokheim’, Proceedings of the PROFES’99 conference, Oulu Finland, June 22-24, 1999

SESSION 2:

Tools and Techniques in Software Process Improvement

Effective Feature Analysis for Tool Selection

G. Antoniol*, G. La Commare**, G. Giraudo**, P. Tonella*
*ITC-Irst, 38050 Povo (Trento) - Italy
**Sodalia SpA, via V. Zambra, 1 - 38100 Trento - Italy

Abstract

The ESSI PIE ITALO project aims at improving the component test phase for object oriented software. The main steps of the project include the selection of a support tool and the set up of an experiment to quantify the effects of its adoption.

Tool selection was performed according to the DESMET [3] guidelines for feature analysis. Since the choice of a good support tool is crucial for the success of the whole project, but the resources for the selection process were limited, the feature analysis was performed so as to be extremely effective, i.e. able to give the maximum discrimination at the minimum cost.

During each step of the feature analysis (feature elicitation, tool assessment and score analysis) several Effective Feature Analysis Strategies (EFAS) were adopted with the purpose of increasing the discrimination between tools and reducing the cost needed to converge to the final choice. This paper reports on that experience and highlights all the lessons learned in terms of acquired EFAS.

1. Introduction

The goal of ITALO (Improvement of the Testing Activities for the Development of Object Oriented Software), the ESSI PIE (Process Improvement Experiment European project n. 27912, is to improve Sodalia[11] Object Oriented (OO) testing phases by adopting automatic tools to complement and help programmers during

[11] Sodalia is an Italian company developing telecommunication software; it is SEI CMM level 3 assessed and ISO 9001 certified.

component testing activities. *Components* are defined as the smallest units implementing and exporting a user recognizable functionality; component testing [2][4], with current practice, has no automatic support ensuring a high defect removal efficiency and requires almost the same effort as the coding phase.

The approach to the component testing process improvement [1] involved the acquisition of automatic tools, the identification of the baseline projects and the set up of an experiment to apply the new testing process. The experiment [5] will allow measuring the benefits obtained in terms of defect removal efficiency and effort spent with respect to the current practice.

Two main areas are expected to be covered by the selected tool(s): Test Management and Test Execution. Tools in the *Test Management* area provide support to test case design, documentation, maintenance and report generation, while *Test Execution* tools provide support to test script generation, test data generation, and automatic result checking.

The evaluation procedure used in the tool selection phase is based on the DESMET method [3] and aims at assessing the features of the tools against the needs of the organization. DESMET is an appealing evaluation method because it is conceived to help conducting an evaluation exercise in an unbiased and reliable way, which is well formalized and not affected by current company beliefs. In addition this method has been successfully applied by several industries, which reported positively on it.

The feature analysis used to select the best support tool was designed and conducted so as o be extremely *effective*, where the *effectiveness* consists of its ability to provide the maximum discrimination at the minimum cost. Since the choice of the testing support tool was crucial, it had to be done with a clear and complete picture of the differences between the alternative choices. On the other side the resources allocated to the selection process were limited, both in terms of the time in which the selection had to be completed, and in terms of the people performing the task. More particularly, two tool assessors were available full time during the selection, while the two available tool users could be involved only for half a day each. Thus the output of the feature analysis was required to be *discriminating*, i.e., able to put in evidence and highlight all existing differences between tools. At the same time the maximum discrimination had to be achieved at the minimum cost.

This paper reports on that experience and the way it could be packaged. In fact, each step of the feature analysis could be made in an effective way by adopting proper Effective Feature Analysis Strategies (EFAS). The main activities performed during feature analysis will be described, together with the related collection of EFAS. They were derived from one single case study (ITALO), but they can be often interpreted in a very general way, so that they can represent valuable pieces of knowledge also for other companies facing similar problems.

The paper is organized as follows: Section 2 introduces the basic concepts of Feature Analysis. Section 3 discusses the EFAS associated to performing Feature Analysis in an iterative way. The strategies that were used during the construction of the feature list are presented in Section 4. Sections 5 and 6 are devoted to the two iterations performed to select the test support tool, and to the related EFAS. The analysis of costs and benefits, used for the final selection, is described in Section 7. Conclusions are drawn in Section 8.

2. Feature analysis primer

The DESMET method, described in [3] was followed in the evaluation of the candidate tools for the ITALO project. The DESMET method is aimed at helping an evaluator in a particular organization in the design and execution of an evaluation exercise, in order to select the best tool in an unbiased and reliable way.

A DESMET evaluation is a comparison among several alternative options, with the purpose of identifying which of the alternatives is best in specific circumstances. Evaluations are context dependent, in that each specific tool is not expected to be the best in all circumstances. An evaluation in one company could result in one tool being identified as superior, but a similar evaluation in another company could lead to a different conclusion.

The DESMET evaluation method separates evaluations that establish measurable effects of using a tool from the evaluations that determine how well a tool fits the needs of an organisation. Quantitative evaluations are based on the benefits expected from a tool, and data are collected to determine if such benefits are actually obtained. A qualitative evaluation, also termed Feature Analysis, is based on identifying the requirements for a given set of activities and mapping them to features that a tool should support.

2.1 Identifying features

The first step of Feature Analysis is the definition of a feature list. The resulting features should be representative of the requirements of all tool users, and should balance technical, economical, cultural and quality aspects. It is often convenient to organize features in a hierarchy, in which features are decomposed into subfeatures, and subfeatures can be in turn decomposed into subsubfeatures.

A good tool should include all the features that are considered the most important for its users. The importance of each feature can be assessed by ranking it in an ordinal scale that could go from *Nice to have* to *Mandatory*. A tool that does not possess a mandatory feature is, by definition, unacceptable.

There are different gradations of *desirability* of a feature, and correspondingly different ordinal scales could be designed. The following ordinal scale was used to assess the importance of each feature in the ITALO project: Mandatory (**M**), Highly desirable (**HD**), Desirable (**D**), Nice to have (**N**).

2.2 Scoring features

Tools are scored against the identified feature list. For this purpose a judgment scale has to be defined, and tool assessors will use it to score the conformance of each candidate tool to each feature.

A simple example of conformance scale, assessing the presence or absence of a feature, is the two value (yes/no) scale. A more refined scale, actually used by ITALO, is given in Table 1 (top), and was derived from [3].

Table 1: *Example of conformance scale (top) and weighting factors(bottom).*

Make things worse	-1
No support	0
Little support	1
Some support	2
Strong support	3
Very strong support	4
Full support	5

M	Mandatory	10
HD	Highly desirable	6
D	Desirable	3
N	Nice to have	1

Individual scores can then be aggregated, provided that weighting factors are defined for the different importance levels of the features. The weighting factors in Table 1 (bottom), taken from [3], were used by ITALO. Even if there is no defined rationale for preferring a set of weighting factors to another one, the weights in Table 1 (bottom) seemed to properly quantify the different degrees of importance of the features. The arbitrariness of this choice suggested to analyze in detail the final results, with reference to the way they are affected by the chosen weights.

For each aggregate feature the weighted sum of the scores is given as a percentage of the maximum achievable evaluation (sum of weights times the maximum score). An overall evaluation is also computed as the weighted sum over all the features, and is still given as a percentage of the maximum achievable score.

The main theoretical problem in computing aggregate scores is that ordinal scale measures are involved in weighted sum calculations. Such relaxation on the dictates of the measurement theory leads to the need of treating the resulting figure of merit with caution. To be sure that aggregate results are not misleading, particular attention should be paid to those high values produced by very high scores on some features and very low scores on other features (see also EFAS 16).

3. Feature analysis iterative approach

When the score sheet is available from completing all feature Analysis steps, it is possible to select the best tool, but in real cases one iteration of Feature Analysis cannot give a high discrimination among the best scoring tools. Therefore an iterative approach is usually adopted, and the initial tool list is successively shortened, according to the outcome of the previous iteration. In this way the risk of choosing a tool on a poor basis is avoided, since from each iteration to the next one the features being assessed, the judgment scale and the evaluation method are refined. On the other side performing several iterations is expensive. Thus an effective Feature Analysis performed iteratively adopts a set of strategies having the purpose of minimizing the number of iterations to converge to the selected tool(s).

The strategies that resulted very effective during the ITALO project are summarized by the following EFAS:

EFAS 1 *Use feedback from previous iteration for new feature list definition.*

EFAS 2 *Use feedback from previous iteration for new assessment method definition.*

EFAS 3 *Prune choices when the related features provide enough discrimination.*

EFAS 4 *Refine those features giving low discrimination.*

EFAS 1 and 2 are related to the use of the feedback from one iteration to the next one. During ITALO the assessment of conformance of the tools to the needed features led to a refinement of the feature list and of the evaluation criteria, that resulted very effective for the next iteration. In particular refining those features giving low discrimination (EFAS 4) allowed a substantial improvement in the next iteration. Of course, a further strategy is to prune as much as possible in the current iteration (EFAS 3).

4. Feature elicitation

When features are being identified, a cost effective procedure has to be adopted. To avoid spending lots of resources in the feature identification activity, proper effectiveness strategies were adopted during the ITALO project. A detailed analysis of the component test process in Sodalia generated several effective features, so that the following EFAS could be abstracted:

EFAS 5 *Analyze process activities to extract needed features.*

Furthermore it should be noted that ITALO was conducted in the context of a process improvement experiment, and therefore the process to be considered is not simply the current one: the improved one has to be anticipated, to allow the choice of the best support tool. A summary of the process analysis performed for ITALO follows.

The *Component Test* is performed to ensure that a given software component satisfies its specifications. It has to be conducted against the design specifications of the component. The tested component can have a size varying from a class to an architectural component.

The following activities represent the main steps of component test:

1 Component test strategy definition
2 Component test design
3 Component test construction
4 Component environment setup
5 Component test execution

The first activity relates to the selection of a proper component test strategy. Such an activity involves managerial decisions that are hardly supported by tools. Nevertheless the choice between black box and white box testing has an impact on the family of tools that will be subsequently used. While test management and execution tools are always needed, coverage or structural testing tools are only optionally selected.

When performing the component test design, the features to be tested are identified. They will be subsequently mapped into single test cases. A tool supporting this activity should allow to link each feature to the associated test case/suite. Thus the documentation facilities should give the feature from which a test case was derived and all test cases associated to it.

Then test cases are built. Several tool features could help this important activity, where test data are defined and support code is written. Possible tool support could relate to documenting the test cases and aggregating them hierarchically. Test case versioning aligned with code should also be supported. When possible, the insertion of existing test cases inside the test suite under construction should be encouraged. Automatic test case generation would be very beneficial. The generation of code for stubs or drivers would also reduce the test case construction effort.

As regards the environment setup, the possibility to specify environment variables or initialize databases would be helpful.

When moving to the execution phase, the tool should enable the definition of a test agenda, i.e., the selection of a group of test cases/suites for sequential execution. The execution of each test suite should also be automated, together with report generation. Furthermore, the tool should also support the manual execution of those test cases which require the user interaction. In cases where run time errors occur, the tool should be able to recover and continue the execution of the test suite. Analysis of the results and reuse of old test cases are included in the test execution activity. A regression check facility would allow rerunning existing test cases. Automatic result check would simplify the corresponding manual work.

In addition to considering the abstract component test process, a collection of real use cases was extremely effective in isolating the needed features. In fact, use cases are instantiations of the test process, according to the actually adopted practices. A detailed analysis of such practices gave indications on the real needs and on the real opportunities of improvement, coming from tool adoption:

EFAS 6 *Collect use cases to identify the needed features.*

Two additional sources that were effectively employed in the production of the feature list are questionnaires (EFAS 7) and expert opinions (EFAS 8). The questionnaires were filled in by the programmers who daily perform component testing. They were asked to describe the way they perform component test, the main difficulties, and the opportunities for automatic support.

EFAS 7 *Define questionnaires to obtain suggestions on the needed features.*

EFAS 8 *Collect expert opinions on the needed features.*

5. First iteration

For the first iteration the 7 following features were exploded into more detailed subfeatures, against which tool conformance was evaluated:

1 Test management and execution
2 Advanced support
3 Interoperability
4 Learnability
5 Economic issues
6 Supplier
7 System requirements

The subfeatures of feature 1 cover the management and execution of the test cases. The subfeatures of the feature 2 are related to the automatic generation of code or scripts supporting the execution of tests. Interoperability (3) requires that the selected tools be compatible with existing tools and can exchange information with them. The tool should be simple to learn and use (4). The tool, training sessions excluded, should cost less than the 27 KECU in the project budget (5). The supplier should be a known reliable company, with affiliates in Italy, better if already an official provider of Sodalia (6). The tool should run with the UNIX (HP and Sun machines) and Windows NT operating systems (7).

For space reasons, only the subfeatures of the first two features are given below:

1	**Test management and execution**	
1.1	[D]	Test case/suite documentation
1.2	[D]	Test case aggregation into test suites
1.3	[HD]	Test suite selection for execution (test agenda)
1.4	[D]	Test suite versioning aligned with code
1.5	[HD]	Test suite automatic execution
1.6	[M]	Test execution report generation
1.7	[D]	Existing test case use when building new test suites
1.8	[D]	Test case execution error handling
1.9	[HD]	Comparison of current and recorded output for regression check
2	**Advanced support**	
2.1	[N]	Generation of support code (stubs, drivers and simulators)
2.2	[N]	Generation of test cases
2.3	[N]	Automatic result check

The tool should have facilities for documenting and labeling the test cases/suites (1.1). Such a documentation can be retrieved on demand and automatically organized in the form of a description report. The individual test cases can be grouped into test suites (1.2) and a test agenda can be defined with the specification of which test suites are selected for execution (1.3). Test suites are under configuration management (1.4) and aligned with the source code. The test suites selected in the test agenda are automatically executed (1.5) and an execution report is accordingly generated (1.6). When defining new test suites, available test cases can be used (1.7). The occurrence of run time errors or exceptions should be handled by the tool so that the error event is traced and the test execution can continue (1.8). Regression check facilities for the automatic comparison of the outcome of previous test suite execution with the current one are also required (1.9).

Automatically generated code and scripts provide the stubs and drivers that replace possible missing classes, set up the environment required for each test case execution and substitute clients or servers that are not available with simulators (2.1). An interesting feature would be the possibility to automatically generate some of the needed test cases (2.2). Code or facilities for the automatic check of the results, given some formal description of the expected behavior, are also useful characteristics (2.3).

Features 1 and 2 could be covered by two different families of tools. In fact tools for test management and execution are typically not expected to provide advanced support facilities (like automatic code generation). For this reason they will be considered two separate categories (category 1 and 2 in the following) for which two families of tools are respectively looked for:

EFAS 9 *Consider different families of tools for different categories of features.*

In the first iteration the score sheet was filled by performing a screening of 9 tools against the selected features. Documentation about the tools was retrieved from the Web and was asked to the suppliers. In order to conclude that a feature is supported by a tool, the following criterion was followed:

if the available documentation explicitly states that the required feature is supported and it describes how it is supported, a tick is added in the score sheet.

There are dubious cases in which a feature is nowhere considered, but could be supported, or a feature is declared as supported in a very generic way, without any technical detail. In such situations it was chosen not to mark the feature as provided. The same criterion was adopted when answers to mails with questions about the tools were evaluated. Generic answers stating that all features are supported were not considered sufficient to mark all features of the tool.

Aggregate scores were not considered in this iteration since the high error intrinsic to the documentation based attribution of a feature to a tool makes them not much reliable. In fact, substantial differences could be due to the particular interpretation that was given to a document or to the lack of specific technical information. A detailed analysis of the score sheet was conducted instead:

EFAS 10 *In the first iteration, do not aggregate scores, perform a detailed analysis.*

As regards the advanced support (category 2), no tool covered in a satisfactory way the area of interest. In fact, the features offered by each tool were very specific and narrowed to a very particular support to automatic testing. Category 2 was no longer included in the feature list for the next iteration. The strategies underlying such a choice are based on the poor coverage (EFAS 11) offered by the tools, which is a somewhat universal strategy, but also on extra information not explicitly coded in the feature list (EFAS 12), about the technical approach to providing advanced support to testing (e.g. generating a class driver which invokes all syntactically available methods apparently covers feature 2.1, but this is done in a way judged not practically useful).

EFAS 11 *Exclude tools with poor coverage.*

EFAS 12 *Use also extra information acquired during the evaluation.*

The application of EFAS 1 allowed the expansion of all features which needed a finer evaluation, and the deletion of features for which the assessment had a high degree of confidence or which were not to be considered any longer (e.g., category 2), since tools were not satisfactory. Addition and merge were also performed to obtain an effective feature list for the second iteration.

The evaluation of specific constraints imposed by some tools on the testing phase and of some specific features allowed excluding some tools from the next iteration. For example ITEX was disregarded since it is tied to the TTCN language for the specification of communication protocols, but Sodalia is not going to use it. Therefore ITEX was not considered in the successive iteration. The following EFAS can thus be derived:

EFAS 13 *Prune choices according to unacceptable constraints emerged during this iteration.*

6. Second iteration

During the second iteration 3 main features were considered:

1 Test management and execution

2 Interoperability
3 Learnability

2 remaining tools, SMARTS and TestExpert, were assessed in this iteration. The other 4 main features from the first iterations were deleted being surely provided homogeneously by both tools or not provided by any of the considered tools.

The 3 features were expanded into subfeatures, and in turn each subfeature into subsubfeatures, for a total of 58 items to be checked. After filling the score sheet, using the conformance scale in Table 1, aggregate scores could be computed and analyzed in detail for the final choice.

The score sheet was filled according to the following procedure. After installing the two tools, for each of them the tutorial was followed to become familiar with the basic functionalities. The user manuals were also studied in this phase and consulted successively. A component of one of Sodalia's previous projects was selected, so that it includes several automatic and manual test cases with the related documentation. The available test suites were manually tried by following the steps described in the testing documentation. Then one automatic test suite and one manual test suite were inserted both into SMARTS and TestExpert. The functionalities of the two tools were experienced while performing the task of inserting, documenting and executing the test cases and of producing execution reports. The use of the tools on a real example taken from a real project was very beneficial for an accurate evaluation of the tools. In fact, many features that had not been deeply investigated during the tutorials were looked for and tried:

EFAS 14 *Exercise the tools on real examples (vs. tutorial).*

Tool users were available only for a very limited time interval. Therefore their contribution had to be maximized by avoiding wasting time in tool configuration or set up. For this reason they were involved only after the successful insertion of the test suites in both tools. Moreover, before involving them, the tool assessors proposed a scoring on which they agreed. Only after the score sheet was completely filled, the result was revised by the tool users (EFAS 15). They had a tutorial presentation of the tools, and then they tried the tools on the tested component. They were asked to say their opinion on the scores given by the assessors for each feature. Their comments were taken into consideration for the final version of the score sheet:

EFAS 15 *To better exploit the limited availability of tool users, involve them only after having configured the tool, set up all in field tests, and filled in the score sheet.*

Scores were then aggregated for subfeatures and for main features. Aggregate scores for features are depicted in Table 2.

Table 2: *Aggregate scores for features.*

Feature	SMARTS	TestExpert
1 Test management and execution	38.2%	71.7%
2 Interoperability	18.0%	81.0%
3 Learnability	93.0%	85.0%
Overall	53.2%	76.8%

TestExpert is definitely better in its support to test management and execution (1) and to interoperability (2). It offers superior facilities for documenting and executing the test cases, and for their versioning. It records the information attached to test cases and to executions in a database which can be accessed by the user to produce customized reports. Explicit support to interoperability is also provided. On the contrary, SMARTS has poor documentation functionalities. It does not use a database and it records only few fixed information, available through a rigid set of reports. Poor support to interoperability is given. SMARTS results to be globally superior in learnability (3). In fact it is simpler to get started with, but in part this is due to the lower number of functionalities and to their lower sophistication level.

Aggregate scores were deeply investigated to avoid errors possibly resulting from the summation of ordinal measures. An example is the global learnability score, discussed above. Thus the following EFAS can be derived:

EFAS 16 *In the second iteration, use with caution aggregate scores and add interpretations to them.*

The underlying technology on which the tool is based is usually reflected in the functions offered to the user. Therefore it is not directly addressed in the feature list, but it underlies several features. An example is the data store used by the two selected tools. TestExpert stores information on test cases in a database, while SMARTS uses plain ASCII files. This affects several features, like e.g. *interoperability*, but it can also be considered as a further element to judge the *internal* features of the tool. When possible such elements should be considered to make feature analysis more effective:

EFAS 17 *Consider the underlying technology as well.*

7. Cost/benefit analysis

A detailed cost/benefit analysis will be available as a result of the three pilot projects in the near future. Nevertheless the pilot projects have to be conducted on one of the two selected tools, so that a preliminary gross grain analysis was performed on the economical impact of the tool adoption, just to exclude one of them. It is not meant to be very accurate, but it is enough deepened to provide some support to include some economical evaluation in the decision making process, in addition to the analysis of the features.

The tool should return the investment in a reasonable time, and, when compared to other tools, it should give a higher net benefit. More formally, if *T* is the annual cost of the testing activity potentially supported by the tool, α gives the predicted saving on the testing activity cost, *n* is the considered life span of the tool in years and *C* is the tool cost, including all needed licenses for *n* years and training, the benefit *B* should be greater than the cost:

$$B = \alpha n T > C \quad \Rightarrow \quad \alpha_{crit} = \frac{C}{nT}$$

Thus a tool with estimated saving α is admissible, i.e. its cost is covered by the benefits obtained during its life span, if:

$$\alpha > \alpha_{crit}$$

Given two admissible tools, their economic impacts can be compared by computing the respective net (gross) benefits:

$$\alpha_1 n T_1 - C_1 > \alpha_2 n T_2 - C_2 \qquad \text{(net benefit comparison)}$$
$$\alpha_1 T_1 > \alpha_2 T_2 \qquad \text{(gross benefit comparison)}$$

Note that the testing cost terms, T_1 and T_2, are different for the two tools, because the tools could impact different sets of activities, with a lower or higher support.

When SMARTS and TestExpert were considered for admissibility, a quick evaluation was sufficient for the first one, while the second one required a deeper investigation. For such a purpose values were accurately assigned to the three terms *C*, *n* and *T*, resulting in α_{crit} = 1%. Since the estimated saving from TestExpert was higher than 1%, the tool was considered admissible.

EFAS 18 *Prune non admissible tools.*

Being both admissible, the one with the highest economic impact and saving on the component testing activities is expected to be the one with the highest feature coverage. In fact, the features offered by TestExpert could give a high support to the component test phase, while the support from SMARTS is more limited. In addition TestExpert could be adopted also during the integration and system test phases. Therefore both α and T are substantially higher for TestExpert. Comparing the net benefits was considered not viable, because of the uncertainty involved in the estimation of the terms in the related equation. Therefore the higher gross benefit of TestExpert and the satisfaction of the admissibility condition were considered sufficient to conclude that the economic impact of tool adoption was in favor of TestExpert:

EFAS 19 *Use gross benefit coupled with admissibility condition when there is uncertainty on the measures of costs and savings.*

Of course such preliminary economic evaluation will be confirmed or contradicted by the future experimentation with three pilot projects. Nevertheless, anticipating such evaluation with the available data can be important to make feature analysis very effective and to avoid the choice of a tool that will not be considered practically adoptable by the whole company for economic reasons.

Finally, the tool selection process itself was analyzed in terms of costs and benefits. The two tool assessors conducted the two iterations of feature analysis during a period of 47 days, including tool installation, configuration, and insertion of all in field tests. Two tool users were involved only for half of a day each. By exploiting the indications in [3], the screening of 9 tools followed by the feature analysis with in field test of 2 tools can be estimated in about 50 to 60 total staff days. Thus the selection for ITALO could be performed with a total cost slightly lower than the minimum predicted by [3]. Therefore the outlined EFAS could actually provide a help in reducing the costs without diminishing the discrimination.

8. Conclusion

In the context of the ESSI PIE ITALO project, a feature analysis was performed to select a tool supporting component test activities. At each feature analysis step all available strategies to make the analysis as effective as possible were used, with the purpose of obtaining the maximum discrimination at the minimum cost. An iterative approach was followed, and the adopted strategies allowed convergence after two iterations.

The effectiveness of the performed feature analysis was analyzed in terms of EFAS, but in general it is also due to the adopted approach, in which a screening, based on available documentation and e-mail answers, was performed against an accurately filled feature list, followed by a test of the remaining tools on real data, with the involvement of the users:

EFAS 20 *Develop an evaluation procedure structured around a screening followed by an in field test on a real example.*

If the individual EFAS are considered, a common characteristic emerges: many of them are very general and can be adapted to different contexts in a simple way. Sodalia had to perform another tool selection after the one for ITALO, and the packaged experience of ITALO could be almost entirely reused in the new situation. Reusing past feature analysis experience is another EFAS (21) that can be adopted only if the acquired knowledge is not dispersed.

EFAS 21 *Reuse effective feature analysis strategies that revealed successful in the past.*

References

[1] V. Basili, G. Caldiera, and D. H. Rombach. *The Experience Factory, Encyclopedia of Software Engineering.* John Wiley and Sons, 1994.

[2] B. Beizer. *Software Testing Techniques, 2nd edition.* International Thomson Computer Press, 1990.

[3] B. Kitchenham. *A method for evaluating Software Engineering methods and tools.* Technical Report TR96-09, DESMET project UK DTI, 1996.

[4] J. D. McGregor and D. A. Sykes. *Object-oriented software development: engineering software for reuse.* New York, Van Nostrand Reinhold, 1992.

[5] S. L. Pfleeger. *Experimental Design and Analysis in Software Engineering.* SIGSOFT NOTES, Parts 1 to 5, 1994 and 1995.

Questionnaire based usability testing

Drs. Erik P.W.M. van Veenendaal CISA
Improve Quality Services / Eindhoven University of Technology
Eindhoven, The Netherlands

Abstract

Usability is an important aspect of software products. However, in practice not much attention is given to this issue during testing. Testers often do not have the knowledge, instruments and/or time available to handle usability. This paper introduces the Software Usability Measurement Inventory (SUMI) testing technique as a possible solution to these problems. SUMI is a rigorously tested and validated method to measure software quality from a user perspective. Using SUMI the usability of a software product or prototype can be evaluated in a consistent and objective manner. The technique is supported by an extensive reference database and embedded in an effective analysis and reporting tool.

SUMI has been applied in practice in a great number of projects. This paper discusses three practical applications. The results, usability improvements, cost and benefits are described. Conclusions are drawn regarding the applicability and the limitations of SUMI for usability testing.

1. A closer look at usability

Several studies have shown that in addition to functionality and reliability, usability is a very important success factor (Nielsen,1993) (MultiSpace,1997). But although it is sometimes possible to test the software extensively in a usability lab environment, in most situations a usability test has to be carried out with minimum resources.

The usability of a product can be tested from mainly two different perspectives “ease-of-use” and “quality-in-use”. Quite often the scope is limited to the first perspective. The ease or comfort during usage is mainly determined by characteristics of the software product itself, such as the user-interface. Within this type of scope usability is part of product quality characteristics. The usability definition of ISO 9126 is an example of this type of perspective:

Usability

the capability of the software to be understood, learned, used and liked by the user, when used under specified condition (ISO 9126-1,1998)

Two techniques that can be carried out at reasonable costs evaluating the usability product quality, are expert reviews and checklists. However, these techniques have the disadvantage that the real stakeholder, e.g. the user, isn't involved.

In a broader scope usability is being determined by using the product in its (operational) environment. The type of users, the tasks to be carried out, physical and social aspects that can be related to the usage of the software products are taken into account. Usability is being defined as "quality-in-use". The usability definition of ISO 9241 is an example of this type of perspective:

Usability

the extent to which a product can be used by specified users to achieve goals with effectiveness, efficiency and satisfaction in a specified context of use (ISO 9241-11,1996)

Clearly these two perspective of usability are not independent. Achieving "quality-in-use" is dependent on meeting criteria for product quality. The interrelationship is shown in figure 1.

the extent to which a product can be used by specified users to achieve goals with effectiveness, efficiency and satisfaction in a specified context of use (ISO 9241-11,1996)

Clearly these two perspective of usability are not independent. Achieving "quality-in-use" is dependent on meeting criteria for product quality. The interrelationship is shown in figure 1.

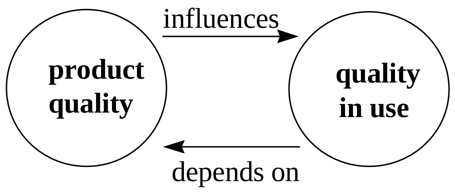


Figure 1 : Relationship between different types of usability.

Establishing test scenarios, for instance based on use cases (Jacobson,1992), can be applied to test usability in accordance with ISO 9241. However, usability testing with specified test cases / scenarios is a big step for most organization and often not even necessary. From a situation where usability is not tested at all one wants a technique that involves users, is reliable but still requires limited resources.

Within the European ESPRIT project MUSiC [ESPRIT 5429] a method has been developed that serves to determine the quality of a software product from a user' perspective. Software Usability Measurement Inventory (SUMI) is a questionnaire based method that has been designed for cost effective usage.

2. What is SUMI?

Software Usability Measurement Inventory (SUMI) is a solution to the recurring problem of measuring users' perception of the usability of software. It provides a valid and reliable method for the comparison of (competing) products and differing versions of the same product, as well as providing diagnostic information for future developments (Kirakowski and Corbett,1993). SUMI consists of a 50-item questionnaire devised in accordance with psychometric practice. Each of the questions is answered with "agree", "undecided" or "disagree". The following sample shows the kind of questions that are asked:

- This software responds too slowly to inputs
- I would recommend this software to my colleagues

- The instructions and prompts are helpful
- I sometimes wonder if I am using the right command
- Working with this software is satisfactory
- The way that system information is presented is clear and understandable
- I think this software is consistent.

The SUMI questionnaire is available in English (UK and US), French, German, Dutch, Spanish, Italian, Greek and Swedish.

SUMI is intended to be administered to a sample of users who have had some experience of using the software to be evaluated. In order to use SUMI reliably a minimum of ten users is recommended based on statistical theory. Based on the answers given and statistical concepts the usability scores are being calculated. Of course SUMI needs a working version of the software before SUMI can be measured. This working version can also be a prototype or a test release.

One of the most important aspects of SUMI has been the development of the standardization database, which now consists of usability profiles of over 2000 different kinds of applications. Basically any kind of application can be evaluated using SUMI as long as it has user input through keyboard or pointing device, display on screen, and some input and output between secondary memory and peripheral devices. When evaluating a product or series of products using SUMI, one may either do a product-against-product comparison, or compare each product against the standardization database, to see how the product that is being rated compares against an average state-of-the-market profile.

SUMI gives a global usability figure and then readings on five subscales:

- *Efficiency*: degree to which the user can achieve the goals of his interaction with the product in a direct and timely manner

- *Affect*: how much the product captures the user's emotional responses
- *Helpfulness*: extent to which the product seems to assist the user
- *Control*: degree to which the user feels he, and not the product, is setting the pace
- *Learnability*: ease with which a user can get started and learn new features of the product.

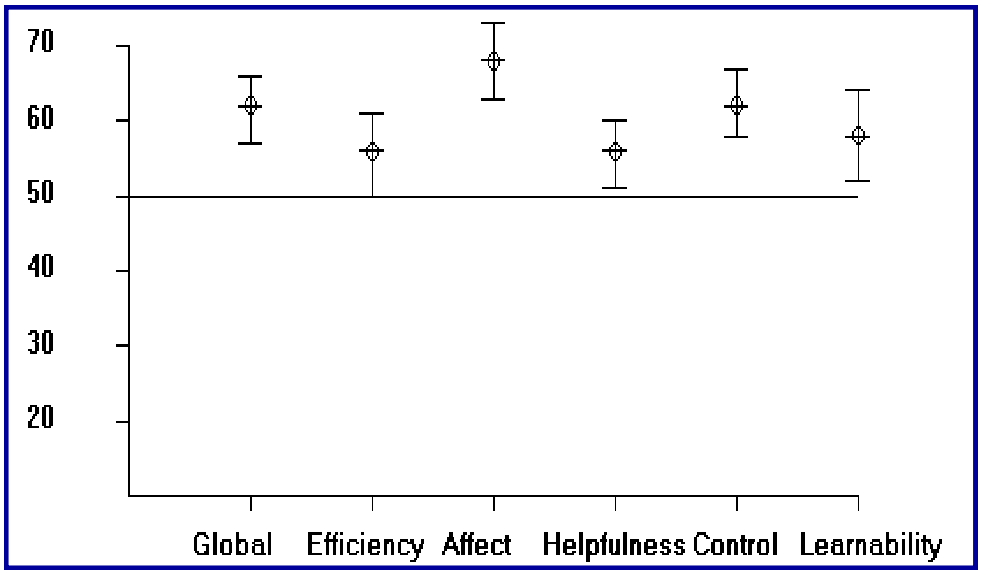


Figure 2: a sample profile showing SUMI scales.

Figure 2 shows an example of SUMI output; it shows the scores of a test and the spreading of these scores (measured by the standard deviation) against the average score of the reference database, reflected by the value 50. Consequently the usability scores shown in the sample profile are positive, e.g. more than state-of-the-art, with a reasonable level of spreading.

SUMI is the only available questionnaire for the assessment of usability of software, which has been developed, validated and standardized on a European wide basis. The SUMI subscales are being referenced in international ISO standards on usability (ISO 9241-10,1994) and software product quality (ISO

9126-2,1997). Product evaluation with SUMI provides a clear and objective measurement of users' view of the suitability of software for their tasks.

This provides a solid basis for specialized versions of SUMI. Recently MUMMS has been developed for MultiMedia products (Measuring Usability of Multi Media Systems).

Any SUMI test must be carried out by asking people that perform realistic, representative tasks. Employing a method such as usability context analysis (NPL,1995) helps identify and specify in a systematic way the characteristics of the users, the tasks they will carry out, and the circumstances of use. Based on the results the various user groups can be described and used to define how these user groups can be represented in the test.

3. Practical Applications

3.1 Project 1: Project Management Package

3.1.1 Approach

Subject to the usability evaluation by means of SUMI was a software package offering project administration and control functionality. The software package is positioned as a multi-project system for controlling the project time, e.g. in terms of scheduling and tracking, and managing the productivity of projects, e.g. in terms of effort and deliverables. The package has been developed by a Dutch software house that specializes in the development of standard software packages.

The SUMI test was part of an acceptance test carried out on behalf of a potential customer. Due to the very high number of users, a number of different user groups, their inexperience with project management software and the great variety of information needs, usability was an important characteristic. It was even looked upon as the critical success factor during implementation. Two main user group were distinguished. One user group was mainly involved in input processing of effort and time spent. For this user group especially operability and efficiency is of great importance. Another user group was characterized as output users. Especially receiving the right management information is important for the output users. Per user group a SUMI test has been carried out.

Regarding the usage of the SUMI technique for the usability evaluation a specific acceptance criteria was applied. SUMI provides quantitative values relating to a number of characteristics that lead to a better understanding of usability. As part of the acceptance test, the SUMI scale was used that provides an overall judgement of usability, the so-called “global scale”. Based on the data in the SUMI database, it can be stated that the global score has an average value of 50 in a normal distribution. This means that by definition for a value exceeding 50 the user satisfaction is higher than average. In the test of the project management package the acceptance criteria applied that for each user group the global scale and the lower limit of the 95% confidence interval must both exceed the value of 50.

3.1.2 Results

The "global scale" regarding both user groups was below the desired 50. For the input user group the score was even a mere 33. The output user group showed a slightly better score. Not only the “global scale” but also most other subscales were scoring below 50.

Because the results did not meet the acceptance criteria that were set a number of usability improvement measures needed to be taken. Examples of measures that were taken based on the results of the SUMI test are:

- extension and adaptation of the user training
- optimization of efficiency for important input functions
- implementation of specific report generation tools for the output user with a clear and understandable user-interface.

3.2 Project 2: PDM system

3.2.1 Approach

At the R&D department of a large copier manufacturer a Product Data Management System (PDMS) is implemented. During the trial phase usability appeared to be an issue and could become a major risk factor during implementation. The time and effort needed to be spent on usability formed a point of discussion between development and the user organization. It was decided to apply SUMI to acquire an insight into the current user perception of the PDMS.

A number of randomly selected users that were involved in the PDMS trail phase were requested to fill out the questionnaire. Twenty six users were selected in this way, of whom twenty-one returned the questionnaire. Six users stated that they didn't use the PDMS often enough. The feedback thus resulted in a 77% response.

3.2.2 Results

The table below shows the overall scores for the various SUMI subscales:

	Global	Efficiency	Affect	Helpfulness	Control	Learnability
Median	36	31	43	36	36	35

Table 1: SUMI scores PDMS.

The various scores are relatively low all round. There didn't seem to be a too large divergence of opinion, except perhaps for learnability. An analysis of the individual user scores did not show any real outlayer (see next table). Two users (one and five) had an outlayer score for one scale (too high). Since it was only on one scale, they were not deleted from the respondent database.

	G	E	A	H	C	L
User 1	60	52	59	**69**	47	32
User 2	57	48	53	62	41	61
User 3	25	19	46	35	22	33
User 4	17	14	28	11	26	23
User 5	61	63	55	44	**60**	64
User 6	24	23	23	36	22	14
User 7	53	62	44	..	..	..
User ..	..	..	..	..	..	..

Table 2: SUMI scores per user.

As stated earlier the various scores were relatively low all round. In general one can say that the user satisfaction regarding the system is too low and corrective action is needed. Some more detailed conclusion were:

- *Efficiency*

  According to the users PDMS doesn’t support the user tasks in an efficient way. One has to carry out too many and too difficult steps. As a consequence one cannot work efficiently and has the opinion that the system is insufficiently customized to their needs.

- *Helpfulness*

  An important conclusion is the fact that the messages are often not clear and understandable; as a consequence the system doesn’t provide much help when one has to solve a problem. The possibilities that the user has in each situation are not clearly shown.

- *Control*

  The user often have the feeling that they are not in control and find it difficult to let the system behave in the way they want it to. They feel save when they only use commands they know. However, they do find it easy to jump from one task to another.

On the basis of the SUMI evaluation it was decided to define a number of follow-up actions:

- a detailed analysis of the problems as being perceived by the users. A number of users is interviewed and asked to explain, by means of practical examples, the answers given to the SUMI questions;

- a study on outstanding change requests and probably increase their priority;

- an improved information service to the users on changed functionality to provide them with more knowledge on how the system operates;

- a re-evaluation of the training material with user representatives;

- a SUMI test was to be carried out on a regular basis (every two/three months) to track the user satisfaction during implementation of the PDMS.

Currently the follow-up is in progress and no new SUMI test has yet taken place. As a consequence nothing can be said regarding the improvement of the usability. However, by means of the SUMI test usability has become a topic within the PDMS project that gets the attention (time and effort) it apparently needs.

3.3 Project 3: Intranet site

3.3.1 Approach

By means of MUMMS, the specialized multimedia version of SUMI, the usability of an intranet site prototype of a large bank was evaluated. The intranet site was set up by the test services department to get well-known and to present themselves to potential customers. The fact that during the test only a prototype version of the intranet site was available meant that some pages were not yet accessible. For MUMMS a special subscale has been introduced, with the objective to measure the users' multimedia "feeling":

- *Excitement*: extent to which end-users feel that they are "drawn into" the world of the multimedia application.

In total ten users (testers) were involved in the MUMMS evaluation. The set of users can be characterized by:

- not having been involved during the development of the intranet site
- potential customers
- four users with internet experience
- six users without internet experience
- varying by age and background (job title).

3.3.2 Results

The table below shows the overall scores for the various MUMMS subscales:

	Affect	Control	Efficiency	Helpfulness	Learnability	Excitement
average score	69	74	62	67	67	68
median	71	77	67	69	67	72
standard deviation	9	12	11	8	6	12

Table 3: Overall MUMMS score table

The various scores were moderately high all round. However, there seems to be a divergence of opinion on the control and excitement scales. Some low scores are pulling down the control and efficiency scales (see next table). Two users from the sample were giving exceptionally low average scores. They were analyzed in detail but no explanation was found.

	A	C	E	H	L	E	Average
User 1	71	81	67	71	74	77	73
User 2	74	74	74	71	67	71	72
User 3	81	84	67	67	74	74	74
User 4	54	51	54	57	64	44	**54**
User 5	71	74	43	58	55	76	63
User 6	64	84	67	81	67	69	72
User 7	51	81	74	54	74	64	66
User 8	71	81	64	74	71	81	73
User 9	77	81	76	84	77	74	78
User 10	64	47	51	57	57	44	**53**

Table 4: MUMMS scores per user.

As stated the usability of the Intranet site was rated moderately high from the users' perspective, although there seemed to be a lot of divergence in the various user opinions. Some more detailed conclusion were:

- *Attractiveness*

The attractiveness score is high (almost 70%). However some users (4, 7 and 10) have a relatively low score. Especially the questions "this MM system is entertaining and fun to use" and "using this MM system is exiting" are answered in different ways. It seems some additional MM features should be added to further improve the attractiveness for all users.

- *Control*

  A very high score for control in general. Again two users can be identified as outlayers (4 and 10) scoring only around 50%, the other scores are around 80%. Problems, if any, in this area could be traced back to the structure of the site.

- *Efficiency*

  The average score on efficiency is the lowest, although still above average. Users need a more time than expected to carry out their task, e.g. find the right information.

On the basis of the MUMMS evaluation it was decided to improve the structure of the internet site and to add a number of features before releasing the site to the users. Currently the update of the intranet site is being carried out. A MUMMS re-evaluation has been planned to quantify the impact of the improvement regarding usability.

4. Applicability of SUMI

On the basis of the test carried out in practice, a number of conclusions have been drawn regarding the applicability of SUMI and MUMMS:

- it is easy to use; not many costs are involved. This applies both to the evaluator and the customer. On average a SUMI test can be carried in approximately 3 days; this includes the time necessary for a limited context analysis and reporting;

- during testing the emphasis is on finding defects, this often results in a negative quality indications. SUMI however, provides an objective opinion;

- the usability score is split into various aspects, making a thorough more detailed evaluation possible (using the various output data);

- MUMMS provides, after detailed analysis and discussion, directions for improvement and directions for further investigation. SUMI can also be used to determine whether a more detailed usability test, e.g. laboratory test, is necessary.

However, also some disadvantages can be distinguished:

- a running version of the system needs to be available; this implies SUMI can only be carried at a relatively late stage of the project;

- the high (minimum of ten) number of users with the same background, that need to fill out the questionnaire. Quite often the implementation or test doesn't involve ten or more users belonging to the same user group;

- the accuracy and level of detail of the findings is limited (this can partly be solved by adding a small number of open question to the SUMI questionnaire).

5. Conclusions

It has been said that a system's end users are *the* experts in using the system to achieve goals and that their voices should be listened to when that system is being evaluated. SUMI does precisely that: it allows quantification of the end users' experience with the software and it encourages the tester to focus in on issues that the end users have difficulty with. Evaluation by experts is also important, but it inevitably considers the system as a collection of software entities.

A questionnaire such as SUMI represents the end result of a lot of effort. The tester get the result of this effort instantly when SUMI is used: the high validity and reliability rates reported for SUMI are due to a large measure to the rigorous and systematic approach adopted in constructing the questionnaire and to the emphasis on industry-based testing during development. However, as with all tools, it is possible to use SUMI both well and badly. Care taken over establishing the context of use, characterizing the end user population, and understanding the tasks for which the system will be used supports sensitive testing and yields valid and useful results in the end.

Literature

Bevan, N. (1997), Quality and usability: a new framework, in: E. van Veenendaal and J. McMullan (eds.), *Achieving Software Product Quality*, Tutein Nolthenius, ‘s Hertogenbosch, The Netherlands

Bos, R. and E.P.W.M. van Veenendaal (1998), For quality of Multimedia systems: The MultiSpace approach (in Dutch), in: *Information Management*, May 1998

ISO/IEC FCD 9126-1 (1998), *Information technology* - *Software product quality* - *Part 1 : Quality model,* International Organization of Standardization

ISO/IEC PDTR 9126-2 (1997), *Information technology* - *Software quality characteristics and metrics* - *Part 2 : External metrics*, International Organization of Standardization

ISO 9421-10 (1994), *Ergonomic Requirements for office work with visual display terminals (VDT's)* - *Part 10 : Dialogue principles*, International Organization of Standardization

ISO 9241-11 (1995), *Ergonomic Requirements for office work with visual display terminals (VDT's)* - *Part 11 : Guidance on usability*, International Organization of Standardization

Jacobson, I. (1992), *Object Oriented Software Engineering; A Use Case Driven Approach*, Addison Wesley, ISBN 0-201-54435-0

Kirakowski, J., The Software Usability Measurement Inventory: Background and Usage, in: *Usability Evaluation in Industry*, Taylor and Francis

Kirakowski, J. and M. Corbett (1993), SUMI: the Software Usability Measurement Inventory, in: *British Journal of Educational Technology*, Vol. 24 No. 3 1993

MultiSpace (1997), *Report on demand oriented survey*, MultiSpace project [ESPRIT 23066]

National Physical Labotory (NPL) (1995), *Usability Context Analysis: A Practical Guide*, version 4.0, NPL Usability Services, UK

Nielsen J. , (1993) *Usability Engineering*, Academic Press

Preece, J. et al, *Human-Computer Interaction*, Addison-Wesley Publishing company

Tienekens, J.J.M. and E.P.W.M. van Veenendaal (1997), *Software Quality from a Business Perspective*, Kluwer Bedrijfsinformatie, Deventer, The Netherlands

Developing a Change Request Management Tool for a Distributed Environment

Horst Lichter

Department of Computer Science

Aachen University of Technology

D-52056 Aachen

lichter@informatik.rwth-aachen.de

Manfred Zeller

ABB Utility Automation GmbH

P.O. Box 10 03 51

D-68128 Mannheim

manfred.zeller@deuta.mail.abb.com

Abstract

This paper presents the experience we obtained in a project aiming at developing and introducing a tool supported systematic approach of change request management. First, we briefly present product and software development at ABB Utility Automation GmbH - Fossil Power Plants and its effort to continuously improve its software engineering capabilities. Then we describe the tool developed to support change management by presenting the basic requirements, its overall architecture, the workflow defined to process change requests and some aspects of the user interface and of tool administration. Afterwards, we discuss the experience we obtained during the project grouped in two categories. The paper concludes by presenting our future plans.

1. Background and Motivation

ABB Utility Automation GmbH - Fossil Power Plants (UTA/F for short) is a company of the worldwide ABB group, employing 800 people, 70 of them in research and development (R&D). The project presented here is focused on the R&D organization. Their business is developing control and monitoring systems for fossil power plants. The products range from controllers to operator stations and engineering systems. Software is a major part of these products. Regarding software development projects two different project types can be distinguished: projects developing basic functionality and components and projects adapting these components according to specific needs of customers. Due to UTA/Fs organization product and software development is distributed over eight sites in five different countries.

Having recognized the impact of good software engineering on software quality as well as on software costs, ABB Kraftwerksleittechnik GmbH, the predecessor organization of UTA/F, started in 1991 an initiative to improve its software process maturity. This process has been continued by UTA/F.

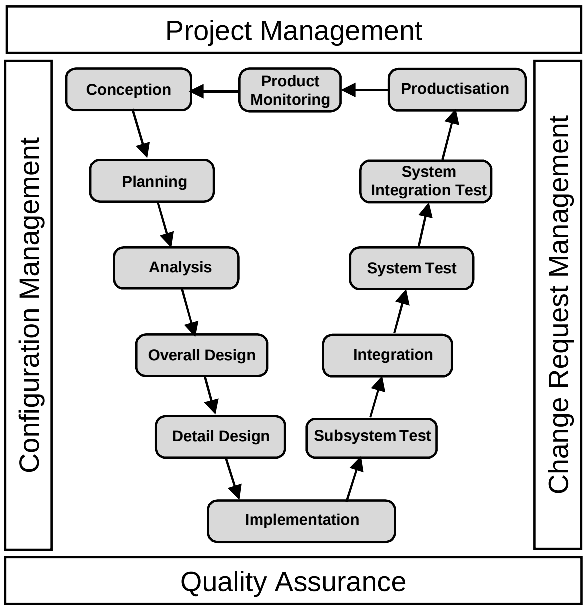


Figure 1: Software process model.

A major and important result of its continuous software process improvement initiative is a software process model which takes into account all major aspects of software projects: e.g. organization, planning, realization and control. The quality assurance is integrated in this process model as well. This model is embedded in UTA/Fs overall product life model. The software process model is based on the traditional phased V-like model described e.g. in Bröhl (1995) and defines the development phases as shown in figure 1. The process model is applied by every development project and is tailored to the specific needs of individual projects. More information on UTA/Fs improvement activities can be found in Lichter (1995) and Welsch (1997).

In this paper we focus on presenting the experience made during the last three years regarding the development and usage of a change request management tool (CRM tool for short). Systematic change request management has been encountered as an important factor to assess product and process quality as well as monitoring quality assurance and maintenance costs. Therefore UTA/F has developed in 1993 a tool supporting change request management activities. This tool was built as a database application using a relational database management system. It offers standard forms at the user interface to enter change requests and to get information about the database content. After having used it for two years the following pitfalls have been encountered:

- Since the system did not support the distributed usage of the CRM database only a small part of the organization (located in Mannheim, Germany) had access to it. Hence the system could not directly be used by engineering organizations of other locations or at plant construction sites.
- As a consequence paper and electronic versions of change requests (CRs) existed in parallel. This lead to several misunderstandings as well as to a CR database content that was never up to date. Hence statistics regarding CR processing did not correspond to reality.
- Since procedure and workflow were hard-coded the tool was rather inflexible. Improvements and adaptations could not be implemented easily. People involved in the CR workflow could not be informed of taking over a CR for further processing via mail.
- The acceptance of the system was low, because it was not available on every workstation and the processing of CRs could not be controlled and monitored by the non-local submitters of CRs.

Based on these findings and due to the meanwhile distributed R&D organization ABB UTA/F launched a project that aimed at introducing a systematic and tool supported CR management that overcomes the problems mentioned before. In the following, we present the main experience and findings of this project. First we give an overview on the most central requirements and describe the overall architecture of the tool together with the CR workflow. Then we present some aspects of the user interface and of tool administration. Finally, we summarize our experience gained so far. In the last section we briefly describe some activities to enhance the current CRM tool that we plan for the future.

2. Essential Requirements

We expected the CRM tool to support the management of detected problems of all UTA/F products. This includes products developed by UTA/F or by a subcontractor as well as involved third party products. Problems may be detected by the R&D organization itself or by any other organization dealing with UTA/F products (test, training, engineering, sales, service, etc.). Therefore the most significant requirements on the CRM tool were:

- worldwide availability within ABB at all UTA/F R&D organizations and at all organizations dealing with UTA/F products
- workflow-controlled processing of CRs
- monitoring the current working state and the history of each CR
- generating CR statistics and reports
- selecting CRs according to online definable criteria
- archiving of fixed CRs
- easy and save worldwide tool administration
- flexibility regarding layout of workflow and user interface
- flexible allocation of persons to the roles defined in the CR workflow
- updating change requests after modification of administration data

After having evaluated some commercial tools supporting CR management we had to realize that no single tool did satisfy our expectations. Either they did not fulfill all of our essential functional requirements or they were not available on ABB's strategic platform. Looking for an individual solution we came into contact with another ABB organization that had already developed a similar application based on Lotus Notes. Due to its experience and because Lotus Notes is a strategic ABB platform we decided to develop an own Lotus Notes based CRM tool. Restricted by the features of Lotus Notes 3.x and by the poor performance of 486er PCs we could not implement all requirements in the first release. Due to the enhancements of Lotus Notes coming with versions 4.0 to 4.5 and being in the meanwhile equipped with powerful Pentium PCs, we were able to gradually implement all our requirements in the succeeding releases of our CRM tool.

3. The Overall Architecture

The central components of the CRM tool are two Lotus Notes databases: the *workflow database* and the *archive database* (see figure 2). The workflow database contains all CRs, the administration data, the request identification numbers and the user's guide. Two types of CRs are distinguished:

- *Public change requests* are CRs on already released products. They are visible to all CRM tool users at any location.
- *Private change requests* are CRs on products being under development. Developers can only create them.

The archive database is used to store closed public CRs in order to relieve the workflow database. Archived CRs are visible to all CRM tool users but can not be changed.

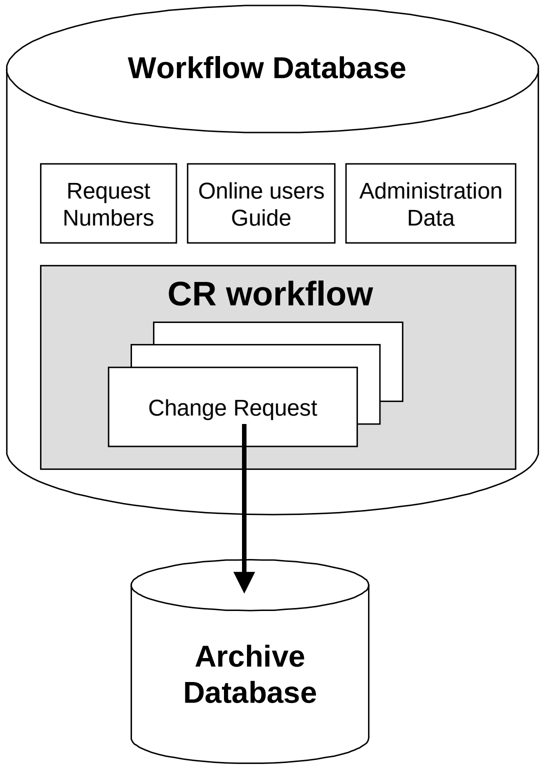


Figure 2: Architecture of the CRM tool.

4. The CR Workflow

In the following we explain the processing of public CRs. Private CRs are processed in a much simpler manner. Public CRs are processed in a controlled workflow. The workflow consists of ten different workflow states as depicted in figure 3. A specific role is assigned to each workflow state. A person assigned to the role of the current workflow state is called *work agent* of the CR. Only a work agent can process the CR and forward it to a succeeding workflow state.

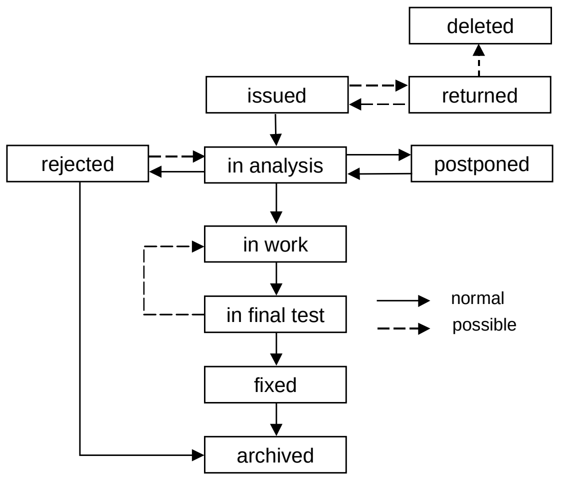


Figure 3: The CR workflow.

Anyone detecting a problem on a product creates a change request using the CRM tool and submits it to the development organization responsible for the product. Then the CR gets the initial state *issued*. Its state changes to *in analysis* as soon as the problem analysis is initiated by the development organization. It remains in this state until the decision is made how to handle the problem. A CR may be *returned* to its submitter, if more information is required for analysis or if the CR is obsolete and should be *deleted* (the submitter only can delete a CR). In case the CR is accepted a developer is responsible to eliminate the problem. During this period the CR keeps the status *in work*. During the succeeding final test the CR is marked by the state *in final test*. The state *fixed* indicates that the final test has been successfully finished. After a non-successful final test the CR is reset into the state *in work*.

In case a non-critical CR has been accepted by the development organization but no resources (manpower, budget) are currently available it may be put into the state *postponed* for later implementation. A not accepted change request is put into the status *rejected*, e.g. if the reported problem is unfounded or if a workaround for the problem exists. Rejected and fixed change requests are transferred into the archive database after a predefined expiration time.

Within the CR workflow there are some situations where people involved in the CR management have to be notified. Examples are:

- When forwarding a CR to a succeeding workflow state the person getting the new work agent is automatically informed by a notification.
- The issuer of a CR automatically gets a notification if the CR leaves the workflow i.e. the CR is postponed, fixed or rejected.
- The development project manager is automatically notified, if the decision authority does not start the problem analysis within a given time after issue.

The CRM tool supports these notifications by automatically generating and sending emails to the corresponding persons.

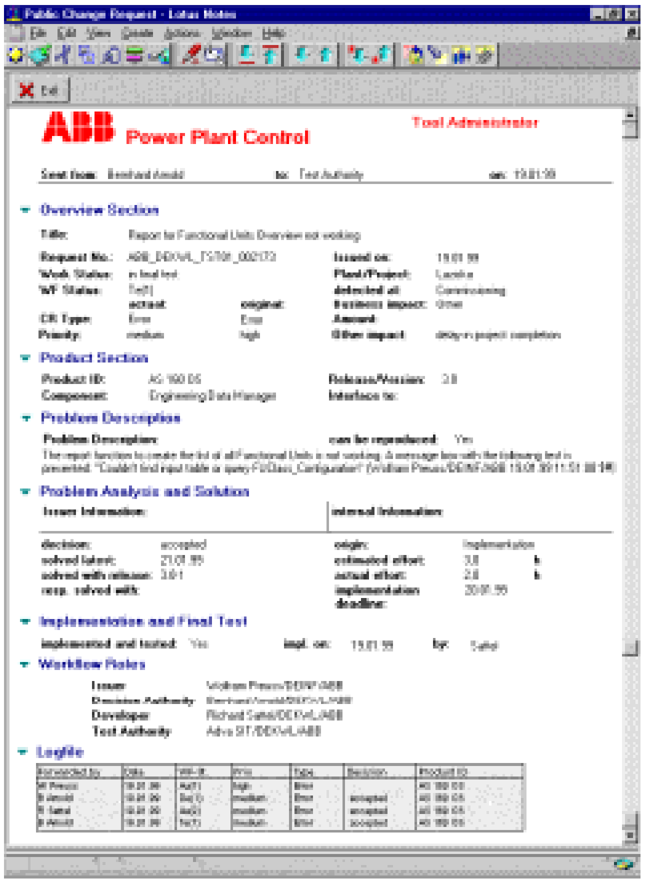
5. User Interface

The user interface of the CRM tool is build by means of forms composed of standard Lotus Notes user interface widgets like text fields, buttons, combo boxes etc.

The layout of the user interface is identical for public, private and archived CRs. The user interface is divided into a number of sections (see figure 4):

- *Overview*: it contains e.g. CR No., title, CR type and priority.
- *Product Identification*: it is identified by product name, release and product component.
- *Problem Description*: this field allows a detailed description of the detected problem. Additional pages (e.g. screen dumps) may be attached.
- *Problem analysis and result*: this section documents the result of R&D's problem analysis as well as decisions that were reached.

- *Workaround and actions taken*: a workaround, if available is documented as well as the actions taken by the individual roles involved in the workflow process.
- *Implementation and test result*: the result of the implementation and of the final test by an independent test group is used for the identification of the succeeding workflow state.
- *Workflow roles*: The persons casting the individual roles within the workflow are indicated. The CRM tool automatically fills in the issuer and the product specific roles like test authority, development project manager and product manager
- *Logfile*: it presents the complete history of the CR. Each step in the workflow is documented with date, forwarding person, etc.


Public Change Request - Lotus Notes
File Edit View Create Actions Window Help
Exit
ABB Power Plant Control
Tool Administrator

Sent from: Bernhard Arnold to: Test Authority on: 19.01.99

Overview Section

Title:	Report for Functional Units Overview not working			
Request No.:	ABB_DEKWL_TS708_002173		Issued on:	15.01.99
Work Status:	in final test		Plant/Project:	Lambia
WF Status:	Te(1)		detected at:	Commissioning
	actual:	original:	Business impact:	Other
CR Type:	Error	Error	Amount:	
Priority:	medium	high	Other impact:	delay in project completion

Product Section

Product ID:	AG 160 DS	Release/Version:	3.0
Component:	Engineering Data Manager	Interface to:	

Problem Description

Problem Description: can be reproduced: Yes

The report function to create the list of all Functional Units is not working. A message box with the following text is presented: "Couldn't find input table or query FUClass_Configuration" (William Preuss/DE/NF/ABB 15.01.99 11:51:08 PM)

Problem Analysis and Solution

Issuer Information:		internal Information:		
decision:	accepted	origin:	Implementation	
solved latest:	21.01.99	estimated effort:	3.0	h
solved with release:	3.0.1	actual effort:	2.0	h
resp. solved with:		implementation deadline:	20.01.99	

Implementation and Final Test

implemented and tested: Yes impl. on: 19.01.99 by: Sattel

Workflow Roles

Issuer	William Preuss/DE/NF/ABB
Decision Authority	Bernhard Arnold/DE/KWL/ABB
Developer	Richard Sattel/DE/KWL/ABB
Test Authority	Adiva SIT/DE/KWL/ABB

Logfile

Forwarded by	Date	WF-St	Prio	Type	Decision	Product ID
W Preuss	19.01.99	Ae(1)	high	Error		AG 160 DS
B Arnold	19.01.99	De(1)	medium	Error	accepted	AG 160 DS
R Sattel	19.01.99	Ae(2)	medium	Error	accepted	AG 160 DS
B Arnold	19.01.99	Te(1)	medium	Error	accepted	AG 160 DS

Figure 4: User Interface

Three field types are distinguished in the user interface: mandatory, optional and computed fields. The type of each field is indicated by the color of the field title. Depending on the value of mandatory fields some optional fields may become mandatory too (e.g. if the value of field *decision* = accepted the optional field *solved with release* becomes mandatory).

Because Lotus Notes does not offer features to create and format statistics we were forced to develop two MS Excel based applications to generate CR statistics and reports. A predefined set of selection criteria can be used to limit statistics or reports to a subset of all existing CRs (e.g. a report may be restricted on CRs of one specific product).

The statistics application generates time related and product related charts and tables with the workflow state as parameter. Standard diagram types are presented that can be changed by the user however.

The report application generates very compact reports only containing the most important data for e.g. analysis and decision meetings.

6. Administration of the CRM Tool

One major requirement was adaptability as well as good support of tool administration. In the following we list some features of the CRM tool regarding these aspects.

Adaptability of workflow and user interface

The workflow is defined by workflow states. The work agent roles and the possible successor workflow states are assigned to each workflow state. Hence, the workflow can easily be adapted by changing these assignments if necessary. The user interface consists of fields. Each field is defined by a field name, its position in the layout and its attributes. The layout of the user interface can be adapted to new requirements caused e.g. by workflow modifications by changing the affected fields and their attributes.

Update of administration data

Changes of administration data (e.g. keywords, workflow definition, field attributes etc.) normally affect existing CRs. To automatically update the workflow and archive database appropriate macros are available ensuring the consistency of the databases.

Distributed Tool administration

Any ABB company worldwide can be connected to the change request management process by installation of the CRM tool on their local Lotus Notes server. Distributed tool administration facilitates the worldwide operation and maintenance of the CRM tool. It is organized through a two level tool administration concept: The global tool administrator normally residing at the main location of the R&D organization manages tool development and maintenance, tool configuration and worldwide tool installation. He also maintains the global administration data. Each local tool administrator manages the local administration data regarding the ABB company he is responsible for.

7. Experience and Lessons Learnt

In this section we present our experience and the lessons we have learned. We summarize the experience regarding tool development as well as usage of the tool in our company.

But first we list some figures giving a better impression on the tool usage.

- Currently the CRM tool is installed on eight Lotus Notes servers in five countries.
- The database contains 2900 CRs (1750 public CRs and 1150 private CRs).
- The size of the database is about 54 MB (18 Kbytes/CR in average)
- The database is replicated between the servers every four hours.
- Currently about 800 users are connected to the CRM tool, 220 users have issuer rights.

7.1 Tool Development

We have chosen Lotus Notes as development platform, because we expected to have many advantages in development, maintenance and administration of a distributed CRM tool. On the other side we knew that there were some constraints of Lotus Notes that we had to deal with.

Restricted by constraints of Lotus Notes 3 and by the poor performance of the target machines, we first developed a CRM tool only implementing some of our essential requirements. In order to get an application with acceptable performance, we decided to implement for each workflow state a corresponding input form with specifically defined access rights on individual fields. This design results in a number of hard-coded forms with poor flexibility. Adaptations regarding workflow or form layout as well as error correction were very costly.

After being equipped with fast Pentium PCs and having Lotus Notus 4.x available we could redesign our application to improve its flexibility and maintainability. This was mainly reached by using only one input form, by centralizing the administration data and by online computing access rights and specific attribute values of input fields. Furthermore we could implement step by step all requirements without too much loss of performance.

There are two major disadvantages of our implemented solution coming from Lotus Notes constraints:

- All data must be part of each CR (viewed as a Lotus Notes document) in order to select and present CRs in views categorized and sorted by arbitrary field value. Splitting up the data and using shared data by referencing to other Lotus Notes documents can not be used. This results in heavy weight CR documents.
- After changing product or workflow specific administration data all affected CRs must be updated. Additional development and test effort was necessary to develop adequate update macros.

In summary we don’t regret having chosen Lotus Notes as development platform especially because ABB regards Lotus Notes as one of its strategic platforms.

Therefore Lotus Notes is installed at each ABB company and its administration has been established worldwide. This facilitates new installations of the CRM tool. The replication of the CR databases is ensured by the existing Lotus Notes infrastructure resulting in a simple procedure to upgrade all locations when a new release of the CRM tool is available: a new release of the tool is installed at the main location by the tool administrator. After one replication cycle it is available at all other locations.

7.2 Tool Usage

In order to systematically introduce the new CRM tool several courses were organized aiming at presenting the central concepts of CR management as well as the main features of the new tool.

In the beginning there was some skepticism on the usefulness of the concepts and the tool support. Particularly people argued against the visibility of public CRs. Since public CRs are visible worldwide, the usage of the CRM tool leads to an overall transparency concerning product related CRs. This situation was pretty new for most people and they needed time to accustom to this situation. In the meanwhile this is accepted and people now regard the worldwide visibility of public CRs as a main advantage of the new CR management process.

At the beginning private CRs were not used frequently. Although private CRs are only visible within a development team and their processing is not controlled by workflow, developers did not see benefits in issuing private CRs. In the meanwhile private CRs are used more and more, mainly to describe current problems or to describe new ideas and future product features. Hence, private CRs are currently also used to collect requirements for subsequent product releases.

Another aspect we like to mention here regards CR management and its results as a reasonable basis for overall product and project management decisions. Due to systematically collecting and processing CRs we now have data available that can be used to assess product quality as well as some aspects of the development process quality. For example we are able to measure the time ellaped between issuing a CR and its analysis (called A-time) as well as the time between issuing and fixing a CR (called F-time). Based on these data we have realized that the A-time has come down from several weeks to about two weeks in average. For CRs classified as critical the A-time is about one week. Before using the CRM-tool the F-time of CRs could not be measured. By means of our tool the F-time is collected systematically for all CRs. Since its introduction this time is decreasing continuously (six weeks in average). Although we use the report facility of the CRM-tool for the monthly quality report, we do not systematically evaluate all the data, but we plan to do more data analysis in future. For this we have started to set up a company wide metric program (see Wünsche 1998) defining basic product and process metrics and especially quality and error metrics.

Besides the usefulness of the concepts of CR management the degree of usability of the CRM tool was an important point concerning its acceptance by the users. Since different user groups, e.g. product managers, department managers or developers are using the tool it has to be customizable to the specific needs of these groups. The feature of Lotus Notes to define private views on a database is used to customize the user interface of the tool. Hence, a user can work with predefined selection facilities of the tool or can define private views presenting exactly those information he is interested in (e.g. product specific CRs, workflow state specific CRs, CRs sorted by work agent etc.). Because Lotus Notes is a strategic platform within ABB most CRM tool users had experience in creating private views and using Lotus Notes applications. This facilities the introduction and the acceptance of our tool.

8. Conclusions and Outlook

Before we present some ideas and features we plan to realize in future versions of the CRM tool we would like to summarize our discussion by comparing the pros and cons of our approach for systematically managing CRs.

We regard the following aspects as the main benefits of our solution:

- Due to the underlying features of Lotus Notes we were easily able to set up a CRM tool that runs in a highly distributed environment. This supports distributed product development as well as distributed product responsibility.

- As a consequence, a CR now can be issued at any location where a problem is detected.
- CRs are forwarded fast and secure to the location responsible for handling the CRs.
- Because CR processing is under control of a defined workflow, we are able to monitor CR processing. This makes it easy to detect those CRs that stay unacceptable long in a certain workflow state.
- Because the user interface only allows a predefined set of terms to characterize a CR, all CRs are describe in a uniform manner (facilitating selection and grouping of CRs).
- Often new requirements could be implemented quickly because both the user interface and the CR workflow were designed to be adaptable.

Of course, there are weak aspects too. We see the following ones:

- The degree of adaptability of the user interface that we have implemented leads to a slower performance of the tool when it starts up.
- Because Lotus Notes does not offer typical database mechanisms like locking, we had to define administrational workarounds to prohibit parallel write access on CRs.
- Lotus Notes does not completely offer the standard Windows interface widgets. Hence, the user interface does not have the standard Windows look and feel.

The experience obtained in the project described in this paper was prevailing positive. We were successful in both, developing a useable workflow based CRM tool and in introducing it in a distributed environment. Nevertheless, there is a lot to do to get all potential benefits from systematically collecting and processing CRs. In future we plan to work on the following topics:

- Integrating our report and statistics application implemented by means of MS Excel in the Lotus Notes based CRM tool (e.g. by an OLE interface).
- Defining and implementing an interface to the configuration and version management tool used by the developers. This is a prerequisite to combine CR processing data and corresponding version management data.

- As mentioned before we plan to systematically evaluate and assess the data concerning CR processing. This needs a deeper definition of the metrics we want to apply.
- Last but not least we plan to make the CRM tool accessible by internet.

References

Bröhl, A.-P., W. Dröschel (1995): Das V-Modell- Der Standard in der Software-Entwicklung mit Praxisleitfaden, Oldenbourg Verlag.

C. Welsch, H. Lichter (1997): Software Process Improvement at ABB – Commonn Issues and Lessons Learnt, Proceedings of Software Quality Management SQM 97, Bath UK.

Lichter, H, C. Welsch, M. Zeller (1995): Software Process Improvement at ABB Kraftwerks-leittechnik GmbH, In P.Elzer, R.Richter (eds.) Proceedings of MSP'95 Experiences with the Management of Software Projects, Karlsruhe, IFAC Conference Proceedings, Elsevier.

Wünsche, W. (1998): A Metric Program for Product Development (in German). Studienarbeit Nr. 1724, Institut für Informatik, Universität Stuttgart.

SESSION 3:

Software Quality

Piloting as a Part of the Process Improvement of Reviews – A Case Study at Nokia Telecommunications Fixed Switching

Janne Kiiskilä
Nokia Telecommunications Oy and Infotech Oulu
Oulu, Finland

Abstract

This article presents some of the results received in a pilot of a review minutes application called IRMA. Based on this pilot, several reasons for supporting the piloting of any planned changes within an organisation can be derived. Some of the reasons are, for example, finding defects, estimating more accurately the training needs, support, costs and benefits of ongoing projects. In addition, a list of suggestions has been provided for other instances intending to plan pilots. It is hoped that this will aid others to successfully manage any changes in their organisations.

1. Introduction

Piloting[12] is discussed in the Software Engineering Institute's (SEI) Software Capability Maturity Model (CMM) Level 5 Key Process Areas (KPAs) Technology Change Management and Process Change Management. It is used as a means of estimating the impact of process improvement and technology change on the software producing unit (Paulk et. al 1993).

[12] I was asked what is the difference between piloting and prototyping. Prototyping is used as a method to elicit requirements during requirements specification. Prototyping can involve for example paper mockups or prototyping tools (Sommerville & Sawyer, 1997). However, a pilot can involve shrink-wrapped software, which can not be modified according to the needs or can only be modified in a very limited fashion. The objective of a pilot is to find out whether or not the software, method or process suits the organisation. The main goal for prototyping is to find out poorly understood requirements.

Estimating change is important, since, typically, the SPU is developing software to meet deadlines, which is more important to the SPU in its imminent future than long term improvement. However, investments in the improvement of the SPU are also crucial in the long run, and sensible weighing of these, sometimes contradicting, goals has to be done. Piloting should be an appropriate aid in achieving this goal. Piloting, when conducted properly, should enable its implementors to envision what kind of benefits the change can really introduce. All the previous estimations are, in this light, only early estimations, and only the piloting will indicate the first realistic results.

Piloting has been effectively used in Nokia Telecommunications (NTC) over a relatively long period of time. It has been observed to be a crucial tool for successfully evaluating and implementing changes within the organisation. This article presents some of the results and valuable lessons learned during the pilot of an in-house review application called IRMA (Inspection and Review Management Application).

The application was originally created for the Base Station Controller (BSC) unit, and was later adopted also by the Fixed Switching (FSC) unit. FSC's project coincided with BSC's project and it was, thus, possible to use their application after some minor modifications had been conducted. The actual application is a Lotus Notes database with an accompanying on-line help database. The application supports a slightly modified version of Fagan's inspection process (Fagan, 1976).

It was observed by the author that the CMM deals with the details of piloting very briefly, which is why it is hoped that this article will shed some more light on the reasons why piloting is so vital when implementing changes within an organisation.

2. Background

After a current-state analysis conducted in the organisation, the introduction of a review management and metrics collection application was seen as the next logical step in the improvement of FSC's review process (Kiiskilä, 1998). After the selection of a suitable application had been conducted, a pilot was arranged according to the company guidelines. The pilot lasted for two months, which started in March 1998 and ended in May 1998. The participants in the pilot project were recruited through a project in which they were participating at the time, and the group consisted of thirty-two people with a variety of job descriptions.

To start with, the participants received training, after which review meetings were held in the usual manner, the only difference being that the review minutes were written using the new application. Once an adequate amount of reviews had been handled using the application in question, the participants of the pilot were asked to fill in a questionnaire concerning the usage of the application. It was observed that during the first training situations some problems occurred. The problem was then traced to an incompatibility between Lotus Notes and Windows NT 3.51 – a combination that did not occur elsewhere than in the PC training class. However, it was observed, unfortunately only at this stage, that this problem, most probably, would have been avoided had a pilot of the training been conducted before the actual training began.

The questionnaire consisted of 24 questions on several aspects of the application and its usage. The questionnaire aimed to measure, for example, the sufficiency of the training, information content, applicability to all reviews, usability, performance, usage of user's guide and on-line helps. Also, participants could comment on the general improvement, i.e benefits, of the application in question, in comparison to the existing method, and whether or not they could recommend taking IRMA into mandatory use thoughout the organisation.

Thirty-two participants returned the questionnaire, thus making the return ratio very good (nearly 100%). One explanation for the high return rate could be the fact that the questionnaires were given to each participant individually. Previous personal experience has shown that when questionnaires are impersonally posted or sent to recipients, the return ratio is relatively low. It must be remembered that employees can be extremely busy, and filling in a questionnaire concerning a project that they do not feel personally involved in, would most probably be an unpleasant task. If, however, they have some level of personal contact with the research or researchers in question they might well be motivated to provide feedback.

3. Pilot Results

The results of the IRMA pilot conducted in FSC cannot be overestimated. In this respect, *the pilot proved to be invaluable*. Throughout the duration of the pilot, numerous improvement suggestions were received and some major defects in the actual program and its usage were located. This aspect of the pilot was not surprising, since IRMA had originally been created for BSC, not FSC, and review practices can vary in different product lines within a company, as was the case in this project. The results of the IRMA pilot are presented and discussed in the following sections.

3.1 Usability

Several questions in the above mentioned questionnaire dealt with the usability of IRMA. The questions were related to main functionality of the application which are writing, approving and locating the review minutes. The results were promising, since the majority of the feedback was positive.

Figure 1 illustrates the results of the pilot users' opinions on how easy it is to write the review minutes using IRMA. The results have some degree of variation, although the majority of the feedback (65 per cent) is, once again, more positive than negative.

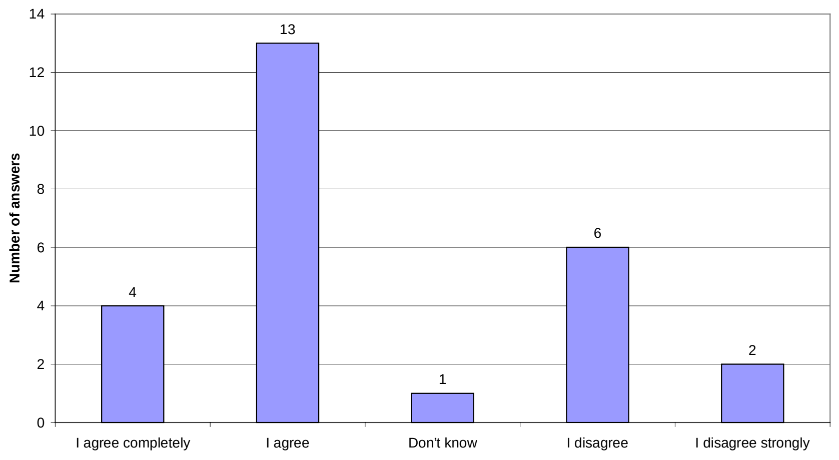


Figure 1. Usability results of writing the review minutes.

On the other hand, the results on whether it is easy to locate the review minutes from IRMA showed a very strong positive trend. The results are illustrated in Figure 2. Locating one particular review minutes can easily become a major problem when the number of review minutes grows from a few hundred to thousands, so this is an important aspect of the application as well.

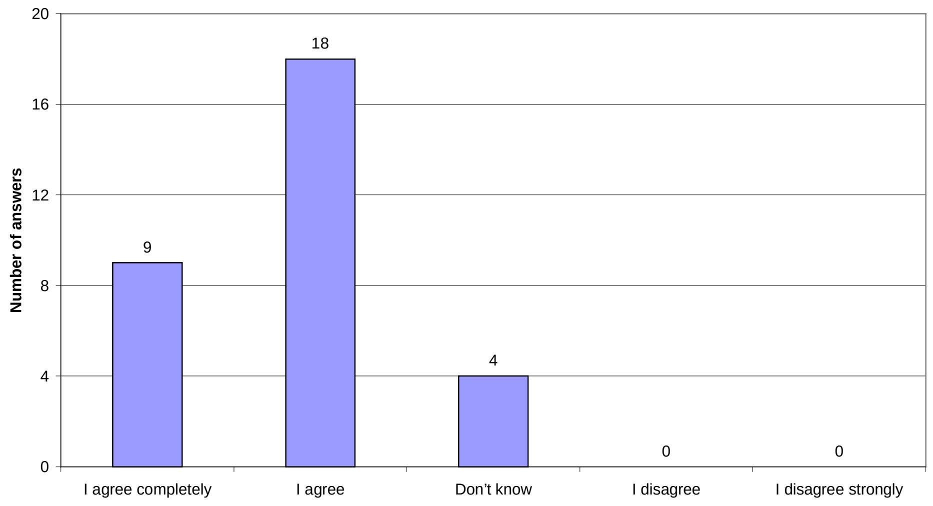


Figure 2. Usability results of approving the review minutes.

In several respects, enquiring about usability with a variety of questions is important as it is relatively difficult to satisfactorily answer one general question on the issue. Also, negative feedback in such a situation would not prove to be very useful, since there is no way of knowing the exact location of the problem. Unless the problem is known, there is no way of fixing it. Usability testing, heuristic methods and other usability evaluation methods would, almost certainly, provide more exact results (Dumas, 1993 and Nielsen, 1994). However, the resources and equipment needed might be too expensive in some situations and, in these cases, piloting might well shed some light on usability issues as well.

A substantial amount of the improvement ideas received from the piloters focused on the usability of the application. *Usability is a very important aspect of the application*, since excellent ideas can be hindered badly by poor usability.

Poor usability frustrates the users and, thus, it will also increase resistance to change.

Usability also affects the piloting and roll-out costs. If the user interface of the application is not self-explanatory, the users will need more training. From the organisations perspective, this is expensive as time spent on training means less time spent doing productive work. There are also other costs related to training, for instance, renting the training location and copying the training material. From the trainer's perspective, training the same thing over and over again can be very strenous.

Usability also affects the amount of support needed by the users. If the application is difficult to use, more error situations are prone to appear. Users will then start calling the application owners and developers and, if the support workload is high, it will hinder other work. Typically, the time spent providing user support is less time spent in developing the tools and processes.

Usability also affects the overall productivity of the users. Human Computer Interaction (HCI) researchers have conducted several studies showing the impact of usability on overall productivity (Preece et. al., 1994).

3.2 Online Help

The IRMA application has both printed and online manuals. The developers were interested to find out which of the two manuals were preferred by users. Perhaps surprisingly, all the participants in the pilot preferred the online help, as is illustrated in Figure 3.

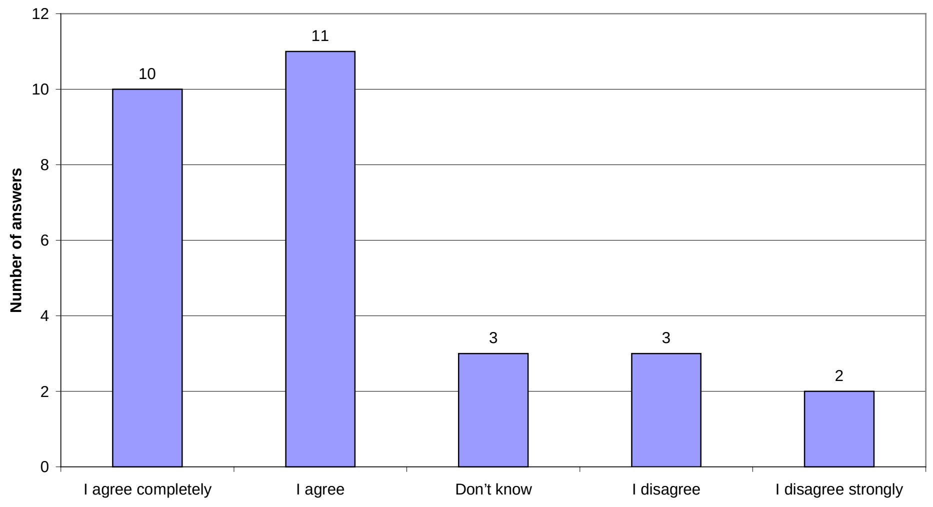


Figure 3. Results of on-line help preference.

However, it was found that users were unwilling to read instructions and help texts, and *more than a half* (57 per cent) of the participants *had not read the instructions at all*. This can also be seen as a clear indicator on the importance of usability – the applications should be usable even without reading the manuals.

3.3 Support for Change

The piloting process is also a method of committing people to change. The users were asked whether or not the IRMA application was considered an improvement in comparison to the previous practices used in writing the review minutes. Previously, the review minutes were written with a word processor, and the files were stored in a VAX-based application called MEMO. The majority of users were preferred the IRMA application, as is illustrated in Figure 4.

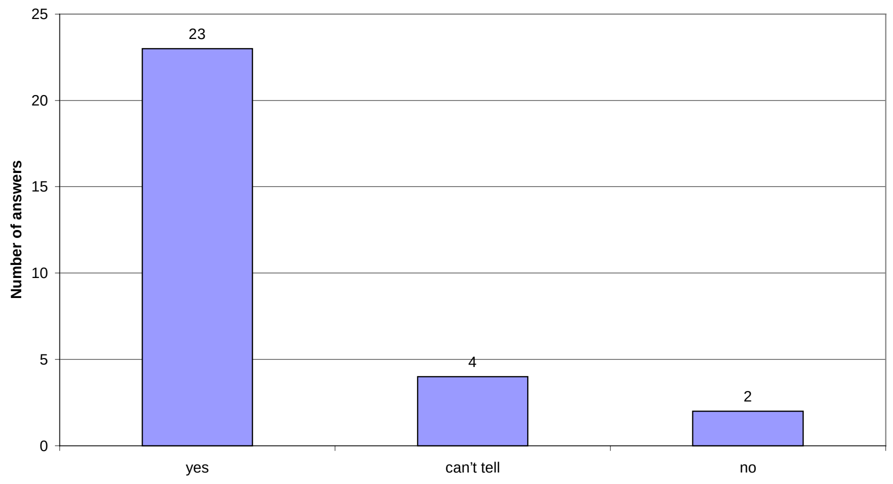


Figure 4. Results for improvement over previous practices.

One additional benefit of piloting is democracy, as users get to decide on the future of the projects. In this IRMA pilot project, users opinions were asked on the future of the IRMA application and on whether or not they recommended the IRMA roll-out for the R&D Management Team. The results are displayed in Figure 5.

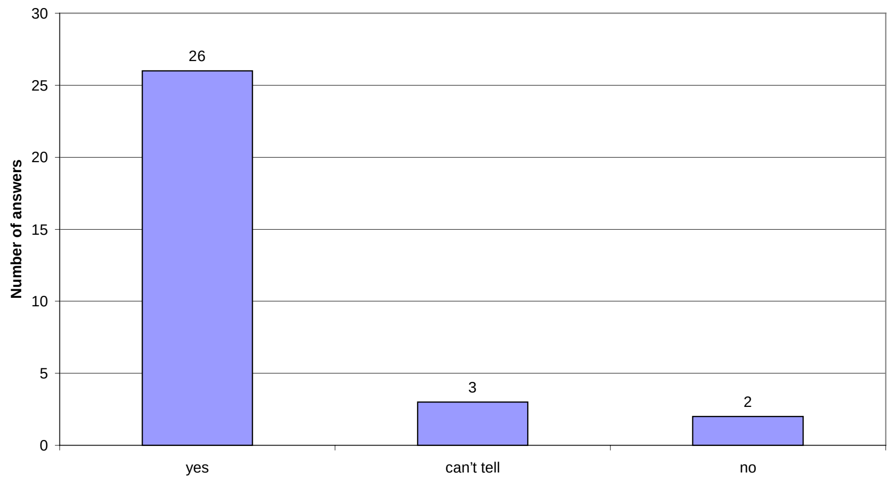


Figure 5. Results on the roll-out recommendation for the R&D Management Team.

The results would seem to indicate that the IRMA application was received favourably by users, and this resulted in a roll-out decision from the R&D Management Team. One of the users with a negative answer to this question stated that the application in itself was all right, but the current server was clearly insufficient. One of the findings in this pilot was the obvious need for a more powerful server, which was later on acquired.

4. Reasons for Piloting

It would seem that piloting is often considered not worth the time or effort. Adding a lengthy piloting phase to the project will mean a longer project, which, in turn, will increase the costs as well. However, in the following sections several reasons for piloting will be outlined.

4.1 Finding Defects

Defects are the enemy of any project. During the IRMA pilot phase many defects in the actual application, the supporting online helps and training were discovered. Some of the defects were fatal, and could have had disasterous consequences should they have been discovered while the application was being used by hundreds of users. The small piloting group ensured that the defects only affected a relatively small group, the disasterous consequences were more easily avoided and remedies communicated.

The users also made lots of improvement suggestions, many of which were implemented rapidly. The visible changes gave the users a sense of power, since they could directly influence the outlook and behaviour of the application. This, in turn, helped commit the users to the change.

4.2 Estimating Training Needs

The piloting phase offers the perfect opportunity to test course material, as well as the length and contents of the course. If feedback on the courses is collected systematically, the courses can be adjusted to better fit the needs of the users.

The feedback form used for this purpose should contain, at least, the following issues (adapted from a course evaluation form used at Nokia):

- Overall estimation of the course and relevancy of contents.
- Course objectives, meeting the objectives.
- Course documentation, presentation material and the actual presententation.
- Course pace and practical arrangements.

The course feedback form should also provide space for general comments, since they often contain valuable information that could help to pinpoint the real problem areas. The comments also often contain direct improvement ideas. The course evaluations should also contain questions in the Likert's scale (or similar), which can be statistically evaluated. In Nokia, these evaluations are used for rewarding the trainers for good performance.

In addition to these benefits, piloting can also help developers to estimate more realistically how many users have to be trained. If the application is easy to use, it will not be necessary to train all users. The differences between individual users can be great: some of the users will learn the application as they use it, others are not so willing to learn by doing. These late adopters can be reached with prescheduled courses that they can attend if they wish. It is also generally considered a good idea to organise a few larger kick-off lectures, supplemented with demonstrations of the application. A large proportion of users will learn the application just by seeing once how it is used. Online helps, manuals and the *frequently asked questions* (FAQ) section can be upgraded based on the questions people ask during the training situations.

4.3 Estimating Support Needs

The piloting phase offers the developers the first real opportunity to estimate the amount of support that the application and its implementation might need. This information is vital for a successful roll-out, since otherwise it can be very difficult to predict the actual support needs.

In order to make a realistic estimate of the expected support requests the application may generate, the number of support requests made by the pilot users must be recorded, and this value then multiplied by the number of users in the actual usage phase. The amount of support requests can be decreased, if:

- The pilot users faced some problems that can be solved and, thus, these same problems will not cause problems in the final user base.

- The problems can be prevented from occurring by enhancing user training, manuals or online help. However, a relatively large percentage of the users do not use the manuals at all, so one must not depend excessively on this (see section 3.2).

If the application is considered critical (large user base, daily usage, failures affect operations), the support has to be extensive. Personal practice has also shown that the more familiar the support is, the easier it is for the users to contact it. *Local power-users* (users specialising in a certain application or method) have been used in Nokia with promising results.

4.4 Estimating Costs and Benefits

The support needs, training costs, licence and hardware costs can be estimated during the pilot in a precise manner. Firstly, the first real, numerical values of these costs must be estimated. Secondly, the first similar results on the actual results of this project – does this really improve our practices – must be obtained? Thirdly, an estimate must be made on how badly the change will interrupt daily routines, and on its possible impacts on the schedules of other projects. It is possible that interruptions to the daily operations could be seen as too severe at that particular time, and a more suitable moment for the roll-out must be decided on.

The costs and benefits will provide the return on investment (ROI) -ratio for this project, and this information is something that is vital in order to be able to make well-informed, objective decisions. The information may also be of help when trying to obtain the necessary support from senior management level (Humphrey 1990).

In the case of the IRMA pilot, it was discovered that a hardware upgrade of the Lotus Notes server was necessary. Identifying this need before the actual roll-out was important, not only for in estimating the related costs, but for the usability of the application.

4.5 Helping the Roll-out Stage

If the project in question promises clear profits, the green light will most probably be given, and the roll-out can take place. Depending on the size of the organisation, this can be a monstrous task that might need phasing. Even in relatively small organisations the roll-out can be consuming, and all possible sources of assistance must be used. The piloting of the project should prepare those concerned with the roll-out in several ways:

- It provides them with a clear understanding of the training needs – who, how many, and how much. In the case of the IRMA application, it was possible to reduce the IRMA Training course from a full day course to a half day course, without decreasing the quality of the course. This, in itself, was a significant saving.

- Knowledge of the most common problems that users will bump into is obtained beforehand. This information can be communicated to the users using newsletters, FAQs, online helps, etc. Or, if the users are directly in contact with the implementers, the answers will be readily available. The application in question has some of the most commonly asked questions, and the answers, in the FAQ-section of the IRMA Online help and WWW pages.

- Concrete results from the pilot, which can be used to convince other users, are obtained. The piloters can be asked to present convincing stories of the improvements they witnessed. We simply used the results in Figure 4 and Figure 5 for this purpose.

- If the piloters were able to influence the project with their requests, they will most probably have a sense of ownership. It is highly likely that they will discuss the project over coffee, and this, in turn, should convince others that the coming change is for the better. In fact, during the IRMA pilot, it was observed that some users started using the application simply because they wanted to give it a try after what they had heard. This sort of feedback can create the so-called “pull” or bottom-up demand for the change, as Gilb & Graham have termed the phenomenon (Gilb & Graham, 1993).

- The piloters also act as a first line support, since the new users will most probably ask questions from the closest source of information they know. This will lessen the support burden of the project implementors.

The above mentioned reasons have been presented in order to assist developers and decision-makers who have to decide whether or not to use a pilot project whenever introducing changes into their organisations.

5. Suggestions for Piloters

Some factors that should be considered before starting a pilot of an improvement project are listed below.

Pilot projects should be carefully pre-selected. Introducing unstable applications or methods might cause unnecessary troubles during the pilot and can cause overly negative response from the piloters. All applications should be properly tested before piloting, and all process changes should be simulated and reviewed. Pilots with no chance of survival are not of benefit to anyone.

Pilot users should be carefully chosen. It is vital to find users with versatile background and job descriptions, so as to ensure better coverage for the different aspects of the project. It should be checked that the pilot users have enough time for the pilot, since if they cannot commit themselves to the pilot, they cannot provide any useful feedback. Also, the number of piloters must be high enough to bear some inevitable losses during the pilot. A number of piloters will change jobs, switch sites and so forth. In the case of this pilot project, a few piloters moved to Nokia oversees sites and were, thus, unavailable for the pilot later on.

The pilot should be carefully tracked, observed and measured. Measurable goals, which confirm the improvement, must be set. The lightest version of this is, for example, a questionnaire, which measures the user’s subjective feeling of improvement in comparison to the previous practices. The pilot users need constant attention, and it is of utmost importance to clear any obstacles they might have and to keep them informed at all times. This will keep the morale up. Support requests (what kind, when and how many) should be kept track of, and the frequently asked questions section should be updated accordingly. If something can be done to the application or the process to prevent support requests, preparations should be made to change those areas, as unnecessary support requests increase costs and could indicate that the usability of the application is poor.

Changes in the project should be prepared for. If the pilot users can suggest improvements and the project can implement them, the users will most probably feel empowered. This could mean that the changes have to be implemented quickly, since the pilots usually have a tight schedule. The developers schedules should be considered, and sufficient time and resources reserved for implementing the possible changes.

In addition to the above mentioned issues, *getting a good start is vital.* A proper kick-off meeting, where the background, goals and expectations of the project are outlined, should be planned. Training courses should be tested and/or piloted in a friendly environment before any pilot personnel are trained. A training course, which falls flat due to unexpected errors (the notorious demo-effect), gives the piloters a very bad first impression which is often hard to shake off.

6. Conclusions

This article presented some of the results obtained from the piloting of an in-house review application called IRMA. The most important result of the pilot was the realisation of the usefulness of piloting. It allowed the developers to refine and improve the application, the related documentation and online help. Several bugs were located, many improvement ideas were created and a large number of them were also implemented during the pilot phase. This created a very positive feeling of ownership for the pilot users, since they were in control of the application development. The pilot users were also asked to do the final assessment on the usefulness and future of the application. The application was found useful and the roll-out of the application was also recommended. Today, the roll-out has proceeded for several months with excellent results.

Piloting was also strongly recommended in this article, and several reasons demonstrating the usefulness of piloting were provided. One of the reasons is, for example, finding defects, which are a constant threat to every project. A pilot should help developers find defects and improvement ideas, as well as refine the application or the method used. This should ease the roll-out, since the users have had a chance to state their opinion and, thus, affect the project. Other reasons include the chance to realistically evaluate the training and support needs, since they greatly affect the costs of the actual roll-out project and operational usage in the future. In this way, the costs and actual benefits of the project can also be estimated with greater accuracy than in the early phases of the project planning. This information enables the management level to make informed and objective decisions on the future of the project.

If a pilot project is in the planning stage, some of the advice in this article may also prove to be useful. It is always advisable to use human resources to do productive work, and this also applies to piloting. The projects to pilot must be chosen carefully, since pilots with no realistic chance of living beyond the piloting stage are usually not worth the time or effort. Also, the pilot users should be carefully chosen: a varied background and several different job descriptions allow for better coverage of the application or method being piloted. Pilot projects also require a considerable amount of attention: the progress of the pilots must be tracked and monitored, and possible obstacles that the pilot users might encounter must be eliminated. Developers and implementors should maintain a flexible attitude and be prepared to change the project according to suggestions made by the users. Change request made by users are usually valid and useful, and the necessary resources in order to implement for the rapid changes must be made available. Getting a good start to the pilot project is vital, as first impressions are hard to shake off.

The improvement of Fixed Switching's review process does not end with this article – this is merely the beginning. Long-term plans covering the general improvement of the in-house review practices have been made. Also, the rigorous use of review metrics to estimate product quality and to track the profitability and efficiency of our review process has been set as an aim. In addition to these, the possibility to embed defect prevention process methods to the review process (see Gilb & Graham 1993 for details) is undergoing investigation. These tasks will inevitably raise an abundance of questions worth future research. Currently the roll-out of the IRMA application is well on its way in FSC, and it is hoped that some of the insights gained during this stage will be published in the near future.

7. Acknowledgements

I would like to thank all the participants of the IRMA pilot project for their feedback and support. I would also like to thank Jyrki Hokkanen for implementing several change requests, always without any delay, during the pilot project. Joanna Matthan also deserves big thanks for editing this article. In addition, I offer my sincere thanks to all the team members of the Process Management Department of the FSC R&D. Last, but not least, I would like convey my deep gratitude to my wife, Virpi, for her constant support and motivation.

References

Dumas, J. S. & Redish, J. C. 1993. *A Practical Guide to Usability Testing*. Ablex Publishing Corporation. 412 pp. ISBN 0-89391-991-8

Fagan, M. E. 1976. *Design and code inspections to reduce errors in program development*. In IBM Systems Journal, vol. 15, no. 3, 1976, pp. 182-211.

Gilb, T. & Graham, D. 1993. Software Inspection. Addison-Wesley Publishing Company. 471 pp. ISBN 0-201-63181-4

Humphrey, W. S. 1990. *Managing the Software Process*. Addison Wesley Publishing Company. Reprinted with corrections. 494 pp. ISBN 0-201-18095-2

Kiiskilä, J. 1998. *Practical Aspects on the Assessment of a Review Process*. In Euromicro 98, Proceedings of the 24th EUROMICRO conference, volume II. IEEE Computer Society Press, 1998, pp. 867 – 870.

Nielsen, J & Mack, R. L. (eds) 1994. *Usability Inspection Methods*. John Wiley & Sons. 413 pp. ISBN 0-471-01877-5

Paulk, M. C., Weber, C. V., Garcia, S. M., Chrissis, M. and Bush, M.. 1993. *Key Practices of the Capability Maturity Model*, Version 1.1. February 1993, CMU/SEI-93-TR-25. Software Engineering Institute, Carnegie Mellon University, Pittsburgh.

Preece, J. et al. (eds) 1994. *Human-Computer Interaction*. Addison-Wesley Publishing Company. 703 pp. ISBN 0-201-62769-8

Sommerville, I. & Sawyer, P. 1997. *Requirements Engineering – A Good Practice Guide*. John Wiley & Sons. 391 pp. ISBN 0-471-97444-7

Early testing of Embedded Software

Marcello Vergano
IAS s.r.l.
Torino – Italy
Ias@alpcom.it
tel + 39 11 9575155

Abstract

This paper presents the Autosim Process Improvement Experiment, ESSI project n. 27585.

The company produces embedded software for factory automation systems. The software integration phase in factory has always been a delicate phase for each project, as the number of faults found and to be fixed in that phase is unpredictable.

Autosim aims at introducing in-house test of embedded software based on the simulation of its environment in the production facility. The software process is enriched of two new phases, definition of simulation models of the factory, and in house test of the embedded software against the models. It is expected that these new activities will allow to find most integration faults in house, where they can be fixed easily and at lower cost. Therefore integration in factory should become predictable, with shorter duration and effort.

The experiment has two main parts: preparation and experimentation. In the former a simulation tool is selected, know-how about simulation is acquired with training, test procedures are defined and integrated in the quality system. In the latter the new approach is applied to a baseline project (the enlargement of an automated warehouse for electronic components) and the results are compared with the current situation.

1. Background and context

In the following we describe the context of the Autosim PIE: the company in which it is performed, the software process before the experiment, and the phases planned for the experiment.

The company

IAS business is the production of software and design of hardware in the domain of basic automation and integrated supervision. In particular IAS specialises in embedded software applications for material handling.

As regards software, IAS covers all phases of development, from analysis of requirements to installation. Requirements are usually produced by a system integrator, which develops the whole production facility, and interacts with the final customer. The software runs on networks of PCs or PLCs (Programmable Logical Controllers).

As regards hardware, IAS produces the hardware design for PLCs. IAS does not cover hardware implementation, assembly and installation.

IAS has already faced important organizational and technical changes. From a small pool of technicians, IAS has grown to a software company. The competition in a not growing market has increased, while productivity has not increased and the customers' demand for both reliability and cost effectiveness has. IAS has therefore began an activity aimed at increasing the quality and reducing developing costs for software. Introducing better tools and techniques to test the embedded software is an answer to these objectives. Autosim aims at introducing in-house test of embedded software based on the simulation of its environment in the production facility.

IAS is a very small company started by 4 technical people in 1989. In 1992 the company totalled 8 people and was known for its technical competence. But it was still more similar to a pool of associate technicians than to a structured and managed company.

In 1992/3 the first reorganisation occurred, in order to structure the company, define clear roles and establish management procedures. In 1994 the company had staff of 10 and was one of a few survivors to a crude competition that made many similar companies in the region disappear. Given better management practice and project accounting, it was now clear that not all projects had the same profitability and the same quality, and that these two factors were related: basically, lower quality means longer software integration phase and less profit (or loose). The quality issue was raising more and more interest in management, while customers (system integrators) were also asking for more quality and ISO9001 certification by 1998 at latest.

In 1995 it was decided to anticipate customer request, and the second reorganization started, aimed at restructuring the company to introduce an ISO 9001 compliant quality system. Certification was achieved in February 1997.

Today, IAS strengths are

-flexibility and fast response to customer requests

-good technical reputation

-a well designed quality system is in place, effective and not too cumbersome

-established know-how in the material handling domain

The software process in IAS

A typical project in IAS undergoes the following phases: received call for proposal, application analysis in order to to write proposal, proposal, contract signed, analysis, hardware design (=electrical layout design, including PLCs), software design and coding; stand-alone software test (carried out in house), PC/PLC-to-host integration test (in house, host simulated), hardware test (= test of connections to sensors, actuators and electrical connections, carried out in factory), software installation (in factory), software integration test (in factory), documentation writing (electrical layout, code, operator manual and user interface description).

PC software is developed in C or C++ , Oracle, occasionally Assembler. The development environment is MS Windows based Borland or MS development environment. The typical target environment is MS DOS (clients) or Windows NT (servers) with Ethernet network.

The tools used for hardware design are: EPLAN 4.11, AUTOCAD - SPAC. The programming languages used are the PLC proprietary languages of each vendor (Siemens, Telemecanique, Allen/Bradley, Omron, Hitachi, CGE). The same applies for the tools for software design/coding. Control View, Usr, Genesys are used to write supervision software modules.

Projects have different levels of quality. To deliver the same level of quality to the customer, late minute fault fixings at the customer site are often needed. Since these fixings are not payed by the customer, this means that profit is traded off for quality.

Luckily, thanks to the quality system in place, it is quite easy to understand where the problem comes from.

The software integration test in factory does not give satisfactory results in that in the case of unpredictable problems, the phase typically exceeds plans of 100 - 200%. Since other phases exceed at most a reasonable 5-10%, the performance in the software integration phase decides whether an IAS project is profitable or not.

Faults discovered in software integration in factory originate from:

- hardware implementation and installation. Hardware design is carried out by IAS, while implementation and installation are done by third parties, contracted by the system integrator, and IAS has no control over this work. Hardware can be inconsistent with the design because of faults introduced by these third parties and because of changes made and not communicated to IAS. The introduction of ISO9001 procedures enabled IAS to trace these faults (and their cost) to those responsible for them.

- software design and coding. These faults are introduced by IAS and could be reduced by using early testing.

The stand alone software in house test has a limited coverage because more testing would require the simulation of the external environment of the embedded software. Recently a customer (system integrator) developed and gave to IAS the simulation models of the external environment. Using these models for thorough in house test, the software integration test in factory, made by a third party company in an overseas factory, found virtually no faults.

The project phases

The experiment consists of a preparation and an experimentation phase. During the preparation phase a new process is defined to encompass simulation activities, a suitable tool is selected, the staff is trained. In the experimentation phase the new process and tool are experimented on a typical IAS project. In parallel the process is modified and tuned as needed.

2. The experiment

We describe here the work performed in the first half of the experiment.

2.1 New phases added

The idea of the experiment is to anticipate the phase in which faults are found to the in house test. Until now this approach was not applied because, in the case of embedded software, extensive test requires that the software be connected with its environment, that is sensors and actuators linking it with the actual factory. As the factory cannot be reconstructed in house, it should be simulated. This is a long and costly activity that has not been performed until now. But today new tools exist that facilitate this activity, and their use and integration in IAS process is the scope of Autosim.

The software process in IAS is modified as follows (activities in italics are new): received call for proposal, application analysis in order to to write proposal,

proposal, contract signed, analysis,   hardware design, *definition of simulation models of factory*, *validation of simulation models*, software design and coding; stand alone software test (in house), *PC/PLC-to-factory test*, using simulation models (in house), PC-to-host integration test (in house), hardware test (in factory), software installation (in factory), software integration test (in factory), documentation writing.

We describe here the activities in italics.

Definition of simulation models of factory

The embedded software receives inputs from sensors in the factory (proximity sensors to recognise a piece or a machining tool is present/absent, weight, pressure, force sensors, etc), processes these inputs and reacts, in function of its state, controlling devices placed in the factory (alarms, engines, valves, ..). The customer's requirements, and analysis and hardware design phases, completely define the environment of each PC or PLC. From these data, a simulation model is built for each PC or PLC. The simulation model replaces the factory environment and can be executed (usually on a PC) in house.

It is particularly important that the simulation model is *not* defined by the person charged of the hardware or software design. This could bias the model and reduce its ability to test the hardware and software design of the embedded software. In other words a specific role charged of defining simulation models will be defined.

Validation of simulation models

The simulation model is now reviewed and executed to verify if it complies with the customer's requirement, with analysis and hardware design. The main verification steps are: verification of the model itself, verification of input and outputs to and from sensors and actuators (number, type, behaviour).

PC/PLC to factory test

The software is tested in house against the simulation model. The PC or PLC is connected, using a dedicated card, to the PC on which the model is defined. The software is executed, the model interacts with it. Faults in the software (and possibly in the simulation model) are found and fixed.

Defining early a simulation model of the factory environment of each PC/PLC has many advantages. The hardware design is verified once more, and faults, if any, can be found. The software can be tested thoroughly, and many faults can be found. All the faults (hardware or software) found here would otherwise be found in the software integration phase, where the cost of removal is much higher.

2.2 Measures

The effect of the changes in the process will be controlled by using the following measures. Measurements already taken are in normal font, measurements to be added are in italics.

To monitor the in house test phase	estimated effort, estimated duration, actual effort, actual duration, number of faults discovered, *type of faults discovered*
To monitor the in house integration test phase	*effort, duration, number of faults discovered, type of faults discovered, simulation models effort (definition and validation)*
To monitor the in factory integration phase	estimated effort, estimated duration, actual effort, actual duration, number of faults discovered, *type of faults discovered*

An analysis of measures on past projects has been made too. Although the data collected is not always completely reliable, some trends have been discovered. Figure 1 plots effort and size per project. A linear relationship seems to apply, also for the largest project considered.

Figure 2 plots density of non conformities versus size. Except for two outliers, density seems to be not related with size, and always below a threshold. Finally, Figure 3 shows density of non conformities versus productivity. Here the plot seems to suggest that projects with higher productivity have also higher defect density. Also, in many cases given the same productivity, defect density varies widely. We will compare measures from the baseline project with these measures from past projects to analyse the effect of simulation based early test.

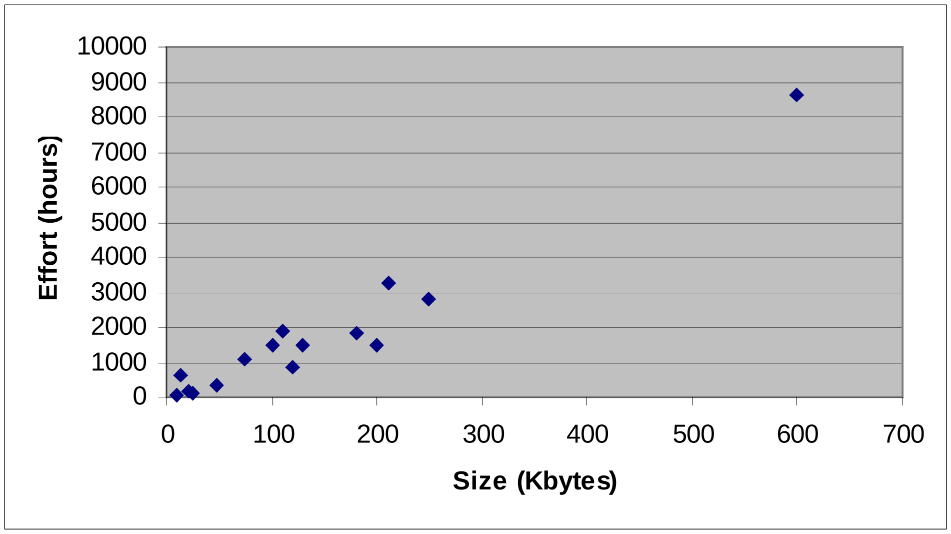


Figure 1 – Effort vs. size for past projects

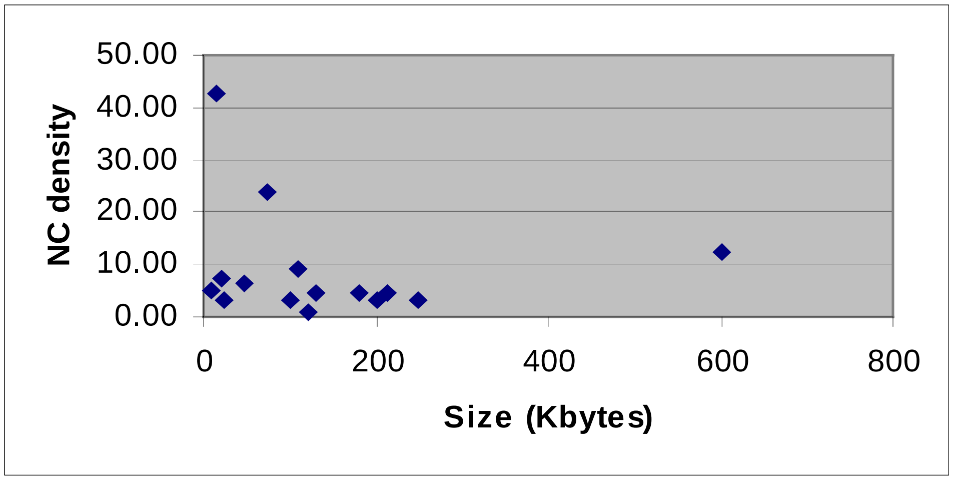


Figure 2- Density of non conformities vs. size

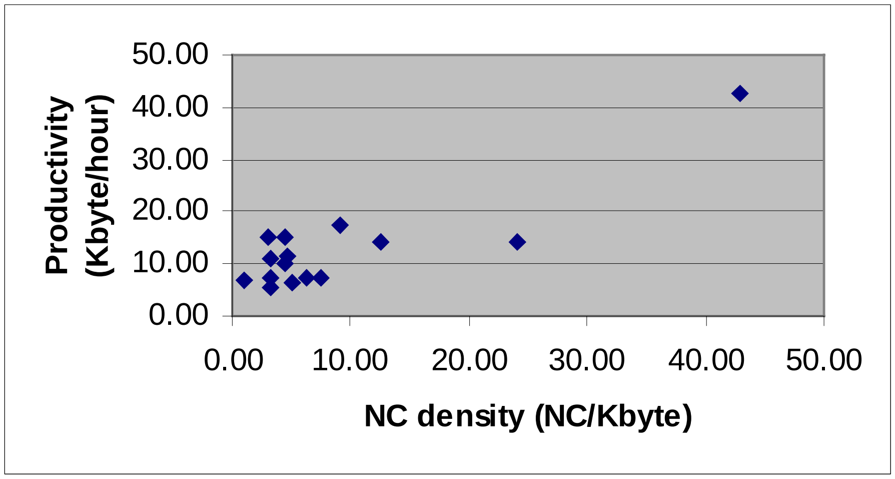


Figure 3- Density of non conformities vs. productivity

2.3 Tools

We have analysed two categories of tools.

- general purpose, lower cost simulation tools with some capabilities of interfacing the external world (i.e. piloting external I/O), such as Witness, Arena, Simula++
- special purpose simulation tools (such as Simac), particularly apt to simulating factory environments and to piloting external I/O

Finally the choice was for Simac, given its more specific capabilities of supporting simulation of embedded systems, and particularly factory automation environments.

The typical configuration of a simulation is shown in Figure 4

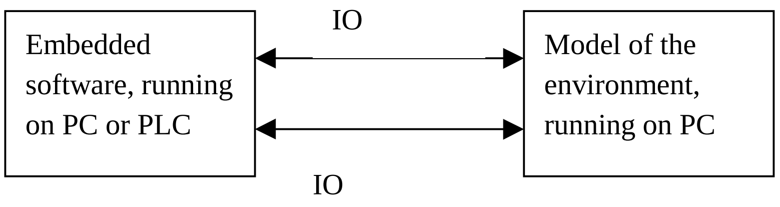

Embedded software, running on PC or PLC
IO
IO
Model of the environment, running on PC

Figure 4 - Embedded software and simulation models

3. Conclusions

The Autosim project is now halfway from conclusion, and experimentation has just started. However, a number of observations can already be made.

- Acquiring the skill to build effective simulation model is a demanding task, the effort to be allocated for it should not be underestimated.

- Building simulation models should not be performed by the same person(s) that build the programs to be tested. The main reason is that the programmer would be tempted to build simulation models that demonstrate his program is correct, instead of building models that find the most defects. Another reason is that building models requires different skills than building the program.

- Currently we have a limited number of models, but it is already clear that a library of models, a librarian charged of their maintenance, and a set of library procedures are needed.

- We are building models on an ad hoc approach, project by project. However experience from past projects suggests that models could be reused easily from project to project, provided that they are designed with reusability in mind. We think that models could be reused even more and with less problems than modules in the applications.

4. Acknowledgements

This work was supported by the European Commission, under contract
Esprit/ESSI Autosim n. 27585.

Capture-Recapture Estimations for Perspective-Based Reading – A Simulated Experiment

Thomas Thelin and Per Runeson
Department of Communication Systems, Lund University
P.O. Box 118, SE-221 00 Lund, Sweden
{thomas.thelin, per.runeson}@tts.lth.se

Abstract

Inspections are established as important contributors to software quality. In order to improve the quality control and the efficiency of inspections, new methods and models are presented. Capture-recapture (CRC) models estimate the number of remaining faults after an inspection. Perspective-based reading (PBR) focuses different inspectors on different areas, thus reducing overlap in terms of detection effort and faults found. The combination of PBR and CRC is, however, assumed to give problems, since the prerequisites of the two techniques are partly contradictory. In order to investigate whether this is a practical problem, a simulated experiment is designed. The experiment results indicate that the CRC estimators are rather robust to simulated PBR data. For three inspectors, Mh-JK and DPM are superior, while for six inspectors, five out of six investigated estimators show acceptable results. The DPM estimator gives acceptable estimates for three inspectors, but not for the six-inspector case. The experiment gives a basic understanding on the relation between PBR and CRC.

1. Introduction

Inspections are gradually growing into state-of-the-practice in software engineering as efficient contributors to improved software quality, and thereby reduced costs [Fagan76, Gilb93]. During the last years, new techniques have been presented which add value to inspections, in terms of better quality control and improved efficiency. Two new techniques are capture-recapture estimates which has its origin in biology [Otis78], offers an opportunity to estimate the remaining number of faults in a document after an inspection [Eick92, Eick93]. The perspective-based reading technique [Basili96] is aimed at providing more efficient inspections by focusing the inspectors on different aspects or (CRC) and perspective-based reading (PBR). The capture-recapture technique, perspectives of the document during the inspection.

After the application of capture-recapture to software engineering was proposed, different studies have been conducted on simulated as well as real data. The purpose has been to understand the behaviour of capture-recapture estimation models for software engineering data, and ultimately to find a superior model [Vander Wiel93, Wohlin95, Briand97, Briand98a, Briand98b, Wohlin98, Runeson98, Petersson99]. Only [Briand98b] analyses the impact of the PBR technique on CRC estimates.

There is a contradiction between the principles of CRC and those of PBR. The aim of PBR is that different inspectors should *not* find the same faults, while the CRC estimators are based on the *existence* of an overlap among faults found by different inspectors. The relation between PBR and CRC was intended to be investigated on real data. However, the difference among the perspectives was not significantly different [Regnell99], hence data for the investigation with PBR characteristics are simulated. A chi-square test [Siegel88, pp. 191-200] is used in [Regnell99] and in this paper to test whether the perspectives detect different faults. The relations between the characteristics of the data are illustrated in Figure 1. The conformance between real and expected data (1) was not possible to show statistically in [Regnell99]. Hence, in this experiment, data with expected characteristics are simulated (2) to produce expected PBR data.

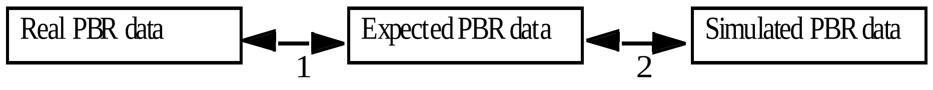


Figure 1. Relations between real, expected and simulated data in PBR inspections.

The results of the study indicate that CRC estimators are rather robust to simulated data with PBR characteristics. For three inspectors, two of the least complex estimators are shown to be superior, while for six inspectors, five out of the six investigated estimators show acceptable estimation results. The experiment gives a basic understanding on the relation between PBR and CRC. In order to validate the assumptions behind the simulated data, the study should be followed-up by experiments with real data.

The paper is outlined as follows. In Section 2, the capture-recapture technique is briefly introduced and in Section 3, the principles behind PBR are presented. Section 4 presents the design of the simulation experiment and Section 5 analyses the results of the experiment. Finally in Section 6, conclusions from the study are presented.

2. Defect Content Estimators

Capture-recapture is a method used to estimate the number of remaining faults in a software document after an inspection. It was first applied to software inspections in [Eick92]. The size of the overlap among the inspectors indicate the number of faults left in a document, and it is used as a basis for the CRC estimation models. The larger overlap among the inspectors the more faults are assumed to remain, and the smaller overlap the less faults are assumed to remain.

A number of different CRC models and estimators exist. Five CRC estimators and one curve fitting estimator are used for simulating the behaviour of defect content estimations in connection with PBR. The six estimators and their models are shown in Table 1, which cover most models that have been applied to software engineering.

Table 1. Defect content models and estimators used in the experiment.

Model / Estimator	**Prerequisites**	**Abbrev./ Reference**
M0 / Maximum-likelihood	All faults have the same probability to be detected by one specific inspector All inspectors have the same detection probability of one specific fault	M0-ML [Otis78]
Mt / Maximum-likelihood	All faults have the same probability to be detected by one specific inspector The inspectors' individual probabilities to detect faults may differ	Mt-ML [Otis78]
Mh / Jack-knife	The faults probability to be detected may differ All inspectors have the same detection probability for one specific fault	Mh-JK [Otis78]
Mh / Chao		Mh-Ch [Chao87]
Mth / Chao	The faults probability to be detected may differ The inspectors' individual probabilities to detect faults may differ	Mth-Ch [Chao92]
Curve fitting / Detection Pro-file Method	The sorted and plotted data should resemble an exponential function	DPM [Wohlin98]

3. Perspective-Based Reading

Perspective-based reading assigns different perspectives to the inspectors to apply when reading. The initial idea behind reading a document from different perspectives is to gain better detection coverage of faults in a software document. This is achieved by focusing the reading effort of different inspectors on different aspects. Basili et al. [Basili96] use designer, tester and user as inspections roles.

Model building is a central part of the three inspection techniques used for PBR. The models used for inspecting a document stem from well-known techniques used in the different software phases. Designers utilise structured analysis, testers utilise equivalence partitioning and users utilise use cases. All these models are structured and adapted to be used as inspection techniques. By combining the three perspectives, the resulting inspections are expected to achieve better fault coverage of the inspected document and less overlap among what different inspectors find. Hence the inspections are expected to be more cost-effective.

The principles behind PBR are, however, contradictory to the CRC estimation models, which are based on the overlap among faults found by different inspectors. If PBR works according to its principles, an inspection according to PBR would provide less overlap among faults found than inspections conducted with ad hoc or checklist reading techniques, given that the same number of faults actually remains after the inspection. The CRC estimators would then estimate a larger number of remaining faults for the PBR case.

4. A Simulated Experiment

4.1 Design and Assumptions

The scope of the experiment is to investigate the estimation performance of six estimators when the fault data input to the estimators have PBR characteristics. In order to investigate the behaviour of the six estimators, inspections with varying characteristics are simulated, generating data correspondingly. All data sets, except one, are generated to simulate PBR inspections. The experiment includes some independent variables which are described below.

- Two different sizes of inspection teams are used. Teams with three inspectors are chosen to reflect an industrial setting. Teams with six inspectors are chosen to investigate whether some estimators change behaviour when more input is provided.

- The number of faults in each simulated document is 30. This variable is chosen based on empirical studies of PBR. Examples of empirical studies using documents that contain about 30 faults are [Basili96, Porter95].

- The number of simulations for each case are 1000 to constitute a sufficient data set for the statistical tests. In Table 2 the 19 simulation cases are shown, where each of the simulation cases include 1000 simulated inspections.

- For every simulation case, three perspectives are used. It is assumed that a document contains three different types of faults, which have different probabilities of being detected. One perspective has high probability ($p \geq 0.6$) to detect one third of the faults and low probability ($p \leq 0.4$) to detect the other two thirds of the faults. In Table 2 the probability levels used for the experiment are presented. For six inspectors it is assumed that two of each perspective are used in each inspection, which means that two high probabilities and four low probabilities are used.

- One combination of probabilities is 0.5, 0.5 and 0.5. These values represent a reference inspection and are not part of the PBR data. The purpose for this simulation is to investigate one simulation where all estimation model's prerequisites are fulfilled. Also, it is used as a reference, which should not be rejected by the chi-square test, see Section 4.2.

Table 2. The probability values for the 19 simulation cases. For six inspectors, two high and two of every low probability values are chosen.

Perspective	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Low	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
Low	0.1	0.1	0.1	0.25	0.25	0.25	0.4	0.4	0.4
High	0.6	0.75	0.9	0.6	0.75	0.9	0.6	0.75	0.9

Perspective	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**
Low	0.25	0.25	0.25	0.25	0.25	0.25	0.4	0.4	0.4	0.5
Low	0.25	0.25	0.25	0.4	0.4	0.4	0.4	0.4	0.4	0.5
High	0.6	0.75	0.9	0.6	0.75	0.9	0.6	0.75	0.9	0.5

The dependent variable is the estimation values of each estimator. This is characterised by its mean and standard deviation values. The relative error is used in the plots to easily compare the results. The definition of relative error used is:

$$\text{Relative Error} = \frac{\text{Estimated number of faults} - \text{Actual number of faults}}{\text{Actual number of faults}}$$

4.2 Analysis and Evaluation

This section discusses the statistical analysis conducted. First the *Kolmogorov-Smirnov test*, *Levene test*, *normal probability plots and residual plots* were performed. The tests and plots aim at investigating the validity of assumptions of normal distribution and equal standard deviation that have to be fulfilled for parametric tests [Montgomery97]. The validation procedure showed that the data do not represent normal distributions. In addition, the data had not equal standard deviation. As a consequence, non-parametric tests are performed in the experiment [Siegel88]. In Table 3 the procedure of statistical testing is listed.

Table 3. The statistical test procedure.

No.	**Test**	**Purpose**
1	Chi-square	Validate the simulation model
2	Kruskal-Wallis	Investigate whether the means are equal
3	Mann-Whitney U	Investigate whether each pair of means are equal

In the Mann-Whitney test (multiple paired tests), a significance level of 0.005 is used and for the other tests a significance level of 0.05 is used.

4.3 Threats

The validity of the results achieved in experiments depends on factors in the experiment settings. Furthermore, different types of validity can be prioritized depending of what conclusions will be drawn from the experiment. In this case, threats to four types of validity are analysed [Cook79]: conclusion validity, internal validity, construct validity and external validity.

Conclusion validity concerns matters on the statistical analysis of results and the composition of subjects. In this experiment, well known statistical techniques are applied and their assumptions are under control. Furthermore, the simulation technique enables generating as many samples as needed, which reduces the general problem in software engineering experimentation of small data sets.

Internal validity is about, for instance, matters that make the subjects act differently in the experiment than they would have done in a non-experimental situation. Again, the simulation approach makes the “subjects” act exactly according to their probabilistic definitions.

Construct validity deals with generalisation of experiment results to concepts or theory behind the experiment. It is assumed that the PBR inspections generate fault detection data with certain characteristics, in terms of detection probabilities. It is assumed that the inspectors have different detection probabilities for different faults, depending on their perspectives. The experiment is conducted on the basis of these expected PBR characteristics. If this assumption is not true, i.e. the PBR inspection data do not show the expected characteristics, the experiment results can not be generalized to real PBR inspections. On the other hand, if the PBR data consistently do not show these characteristics, the PBR inspections do not work as the basic assumptions state, and the value of applying PBR can be questioned.

External validity concerns generalisation of experiment results to other environments than the one in which the study is conducted. Again, as the study is based on simulations, the external validity depends on how close the simulation model is to real environments to which the results are to be generalised.

It can be concluded that the most important threat to the validity is the characteristics of the inspection data. The results should be interpreted in the light of this fact. If real data do not show the expected PBR characteristics, the results can not be generalized to PBR inspections. However, in this case, applying PBR can be questioned as well.

5. Analysis of Estimation Results

5.1 Analysis of PBR Data

Two statistical tests are discussed before further analysis. The tests are number 1 and 2, which are described in Section 4. It turned out to be the same results of these tests for both three and six inspectors.

The result of the chi-square tests was that only one simulation case could not be rejected, namely the simulation with probabilities equal to 0.5 0.5 and 0.5. Hence, the other 18 simulation cases are considered to have PBR characteristics. For future reference of data with PBR characteristics in this paper, only the 18 first simulation cases are considered.

Moreover, the Kruskal-Wallis tests show that H_0 can be rejected for all simulation cases. This means that at least one estimator's mean differ significantly from the others and thus paired test can be performed. The paired tests are discussed during the analyses of mean and standard deviation values.

5.2 Analysis of Teams with Three Inspectors

In Figure 2 and Figure 3, the estimation results of the 19 simulation cases are shown as bar plots of the mean and standard deviation values.

The overlap among faults found for inspectors applying PBR is assumed to be less than the overlap for ad hoc inspections. Hence, PBR is assumed to force capture-recapture estimators to overestimate. Figure 2 shows this assumption to be true in almost all the simulated cases. The larger difference in detection probability and the lower detection probability, the more they overestimate.

A pattern can be seen in all estimators except for Mth-Ch. If the two low probabilities are constant and the high one increases the overestimation also increases. The overestimation decreases when higher detection probability is used. Hence, the more similar inspectors, the smaller overestimates.

Mh-JK and DPM are the estimators which show best estimation results. For all simulation, cases except number 7 to 9, differences are statistically significant (α=0.05). Despite the first three simulation cases where Mh-JK is much better, they both show good estimation results. In some simulations Mh-JK is the best one and vice versa. No pattern is shown when to use DPM or when to use Mh-JK. Therefore both of them is recommended as they show low overestimation and low standard deviation.

Figure 4 and Figure 5 show bar plots of the mean of all the means and the mean of all the standard deviation values of the PBR simulation cases, i.e. the mean of the simulation cases with PBR characteristics (1-18). In order to find an overall best estimator for PBR characteristic data these plots are valuable. They show the overall behaviour of the estimators used in the experiment. The overall and significantly best estimator is Mh-JK using tests and these plots as a reference. DPM is not significantly better than either M0-ML or Mt-ML. Nonetheless, DPM has smaller standard deviation and is hence considered as a better estimator than M0-ML and Mt-ML.

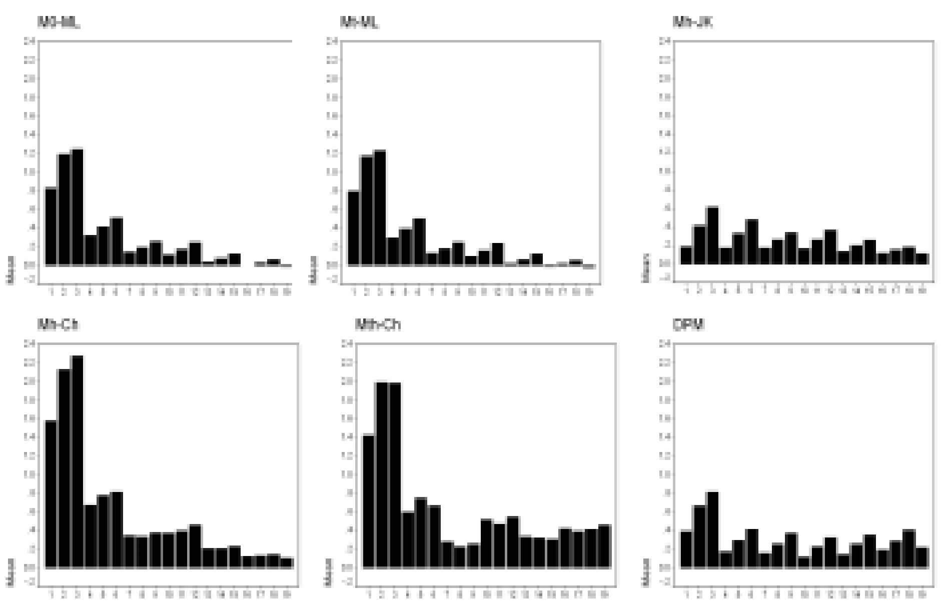


Figure 1. Bar plots of the mean values of relative errors.

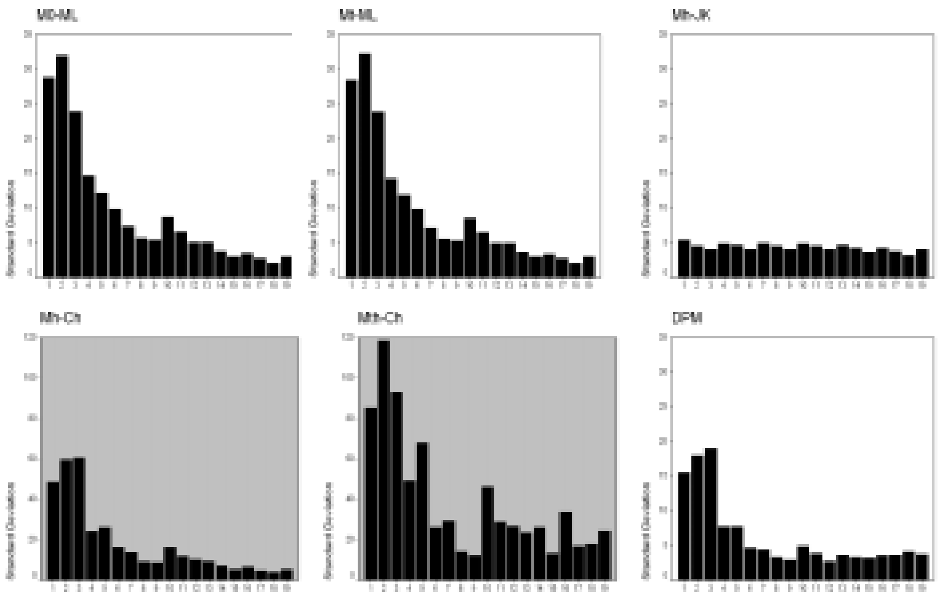


Figure 2. Bar plots of standard deviation values. The numbers below the plots refer to the simulation cases defined in Table 2. Note that the same scales are not used in the plots filled grey.

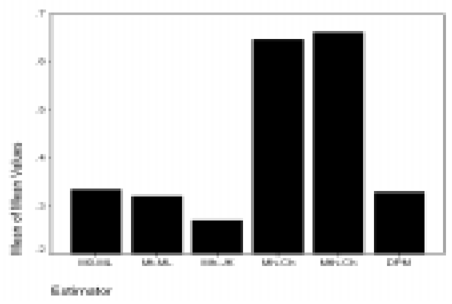


Figure 3. The mean of the absolute values of the means using data with PBR characteristics.

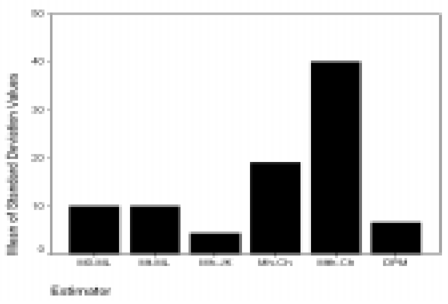


*Figure 4. The mean of the standard deviation values usin*g data with PBR characteristics.

5.3 Analysis of Teams with Six Inspectors

The analysis of six inspectors is carried out in a similar way as for three inspectors. In addition, a comparison between three and six inspectors is discussed. The probability values for the simulation of six inspectors are the same as for three inspectors. This means that there are four inspectors with low detection probability and two inspectors with high detection probability in each simulation case.

In Figure 6 and Figure 7, the estimation results of the 19 simulation cases for six inspectors are shown.

The two best estimators for six inspectors using data with PBR characteristics are M0-ML(and Mt-ML) and Mth-Ch. Both M0-ML and Mth-Ch are significantly better considering all mean values. In all simulation cases, the means of M0-ML and Mth-Ch differ significantly. Although they differ, it is not the same estimator that is superior in all cases. The standard deviation values are small for all estimators except DPM.

For three inspectors a pattern can be seen in several estimators' estimation results. When one inspector's probability increases and the other two remain constant, the estimations become larger, resulting in overestimations. The pattern for six inspectors are reversed; when two inspectors' probabilities increase and four remain constant, the over estimation decreases.

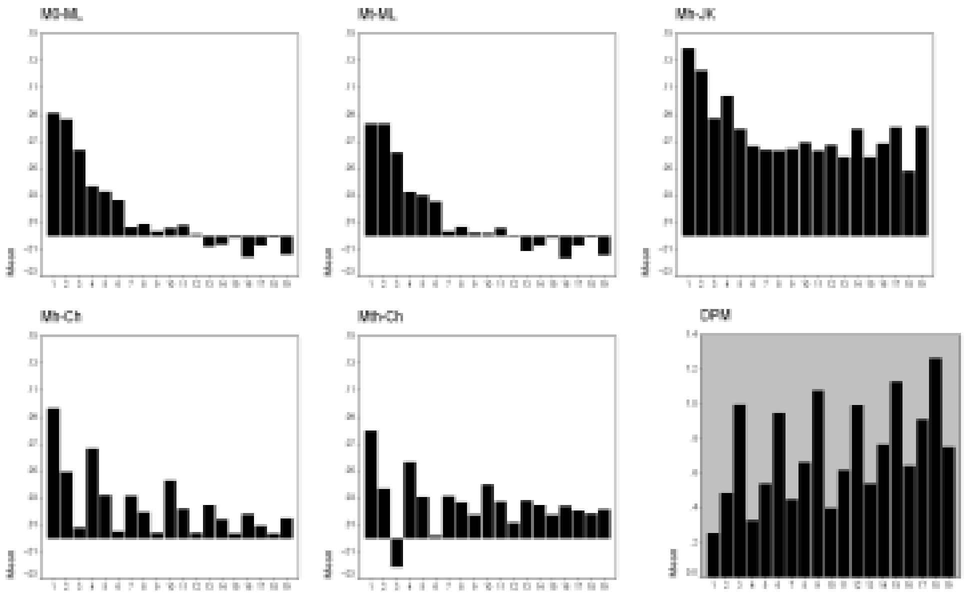


Figure 5. Bar plots of the mean values of relative errors.

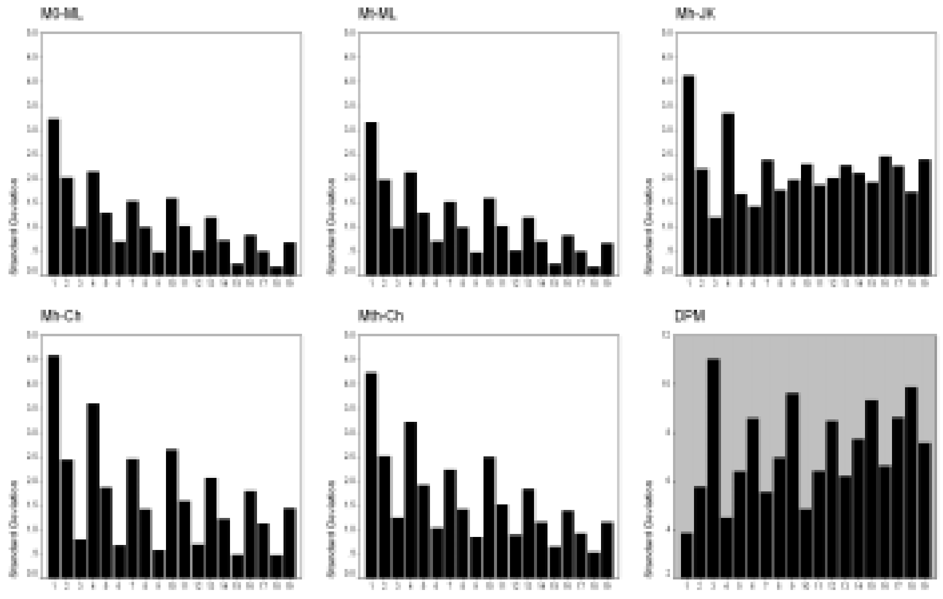


Figure 6. Bar plots of standard deviation values. The numbers below the plots refer to the simulation cases defined in Table 2. Note that the same scale is not used in the plot filled grey.

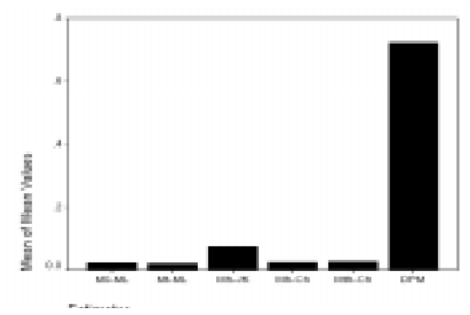


Figure 7. The mean of the absolute values of the means using data with PBR characteristics.

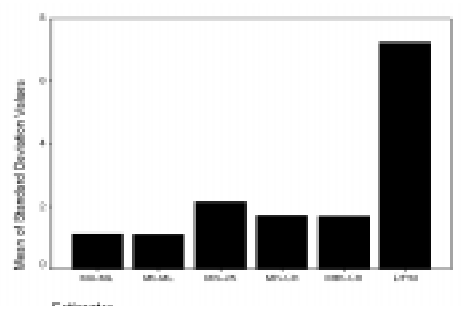


Figure 8. The mean of the standard deviation values using data with PBR characteristics.

Only DPM has the same pattern for three and six inspectors. This may depend on changes in the overlap among the faults that the inspectors detect.

Most of the estimators can be used for data with PBR characteristics. None of the estimators, except DPM, have large bias nor standard deviation. In Figure 7 and Figure 8, the mean of the mean values and the mean of the standard deviation values of the simulations are presented. Considering both mean and standard deviation, M0-ML, Mt-ML and Mth-Ch are the overall best estimators. However, all estimators except DPM estimates accurately.

5.4 Interpretation of the Results

The major conclusion from the analysis is that capture-recapture can be used for PBR inspections. Some of the estimators do not estimate very well for three inspectors, however, that is not the case for ad-hoc inspections either [Braind97, Briand98b].

The model Mth is expected to estimate best, since its prerequisites are less restricted. However, this is not the case. For three inspectors, it estimates rather poorly, on the other hand, for six inspectors, it estimates very good. This behaviour is expected since Mth-Ch needs more data than the other estimators to estimate accurately. The reason behind this may be that Mth has more degrees of freedom and is more complex statistically than the other estimators, thereby Mth-Ch requires a larger amount of input data to perform as well.

Model Mt is the other model expected to estimate well. The estimator in this model, Mt-ML, estimates well using data with PBR characteristics. However, M0-ML estimates at least as well as Mt-ML although it has more restricted prerequisites. Similar results are also discernible in other empirical studies [Briand97, Briand98b, Thelin99]. Since more restricted models often are preferable, this indicates that M0-ML is more appropriate to use for PBR purposes.

For the reference simulation, the models M0 and Mt show best simulation results. This was expected since the prerequisites of all models are fulfilled, and thus a simpler estimator is preferable to utilise.

As a consequence of the analysis, fulfilling the models' prerequisites seem not to improve the estimation result, in particular not for the three-inspector case. This confirms the results in [Thelin99].

DPM estimates very well for three inspectors and very poorly for six inspector. Since DPM does not have similar prerequisites as the CRC estimators it does not behave the same way as these either. The reason may be that when three more inspectors with equal detection probabilities are added, the exponential curve is raised and becomes more flat causing overestimation. As a result, DPM estimates well for data with PBR characteristics for few inspectors and thus can be used as a complement to the CRC estimators.

6. Conclusions

In this paper a simulated experiment of PBR is presented. The main goal is to investigate CRC estimators' behaviour when the input is data with PBR characteristics. In order to judge whether the data have PBR characteristics, a chi-square test is performed. The simulations are performed with both three and six inspectors, since empirical results have shown that the number of inspectors are critical to the performance of CRC estimators [Briand97, Briand98b].

The major conclusions drawn from the experiment are:

- Capture-recapture estimators can be utilised for perspective-based reading. Some of the estimators do not estimate very well for three inspectors, however, that is not the case for ad-hoc inspections either [Braind97, Briand98b]. The detection probability have to be higher in the three-inspector case than the six-inspector to achieve satisfactory results.

- For three inspectors, two of the estimators seem to be superior; these are Mh-JK and DPM. Considering both mean and standard deviation these two estimators are the most robust for data with PBR characteristics. Two of the estimators are not recommended to be used for three inspectors; these are Mh-Ch and Mth-Ch due to their large bias and standard deviation.

- For six inspectors, all estimators except DPM estimate well and are trustworthy. The best estimators are M0-ML, Mt-ML and Mth-Ch. In a former study [Briand97], Mth-Ch has shown to need more inspectors in order to estimate well. This study confirms these results.

- The assumption that CRC estimators overestimate when using PBR data seems to be true in most cases. For three inspectors this is true, however, for six inspectors some of the estimators underestimate in some simulation cases.

Future work with PBR in connection with CRC is to use real data which by the chi-square test is shown to have PBR characteristics. This simulation study should work as a basis for other empirical investigations, from which further conclusions can be drawn concerning CRC and PBR.

7. Acknowledgement

The authors would like to thank Claes Wohlin and Håkan Petersson at the Dept. of Communication Systems, Lund University, and the anonymous reviewers who have carefully reviewed this paper. This work was partly funded by The Swedish National Board for Industrial and Technical Development (NUTEK), grant 1K1P-97-09673.

References

[Basili96]	Basili, V. R., Green, S., Laitenberger, O., Lanubile, F., Shull, F., Sørumgård, S. and Zelkowitz, M. V. “The Empirical Investigation of Perspective-Based Reading” *Empirical Software Engineering: An International Journal*, 1(2):133-164, 1996.
[Briand97]	Briand, L., Emam, K. E., Freimut, B. and Laitenberger, O. “Quantitative Evaluation of Capture-Recapture Models to Control Software Inspections” In *Proc. of the 8:th International Symposium on Software Reliability Engineering*, pp. 234-244, 1997.
[Briand98a]	Briand, L., Emam, K. E. and Freimut, B. “A Comparison and Integration of Capture-Recapture Models and the Detection Profile Method” In *Proc. of the 9:th International Symposium on Software Reliability Engineering*, pp. 32-41, 1998.
[Briand98b]	Briand, L., Emam, K.E. and Freimut, B. “A Comprehensive Evaluation of Capture-Recapture Models for Estimating Software Defect Content” ISERN-98-31, 1998.
[Chao87]	Chao, A. “Estimating the Population Size for Capture-Recapture Data with Unequal Catchability” *Biometrics*, 43:783-791, 1987.
[Chao92]	Chao, A., Lee, S. M. and Jeng, S. L. “Estimating Population Size for Capture-Recapture Data when Capture Probabilities Vary by Time and Individual Animal” *Biometrics*, 48:201-216, 1992.
[Cook79]	Cook, T. D. and Campbell, D. T. *Quasi-Experimentation – Design and Analysis Issues for Field Settings*, Houghton Mifflin Company, 1979.

[Eick92] Eick, S. G., Loader, C. R., Long, M. D., Votta, L. G. and Vander Wiel, S. A. "Estimating Software Fault Content Before Coding" In *Proc. of the 14th International Conference on Software Engineering*, pp. 59-65, 1992.

[Eick93] Eick, S. G., Loader, C. R., Vander Wiel, S. A. and Votta, L. G. "How Many Errors Remain in a Software Design Document after Inspection?" In *Proc. Of the 25th Symposium on the Interface,* Interface Foundation of North America, 1993.

[Fagan76] Fagan, M. E. "Design and Code Inspections to Reduce Errors in Program Development" *IBM System Journal*, 15(3):182-211, 1976.

[Gilb93] Gilb, T. and Graham, D. *Software Inspections*, Addison-Wesley, 1993.

[Montgomery97] Montgomery, D. *Design and Analysis of Experiments,* John Wiley and Sons, USA, 1997.

[Otis78] Otis, D. L., Burnham, K. P., White, G. C. and Anderson, D. R. "Statistical Inference from Capture Data on Closed Animal Populations" *Wildlife Monographs* 62, 1978.

[Petersson99] Petersson, H. and Wohlin, C. "Evaluation of using Capture-Recapture Methods in Software Review Data", Accepted for publication at the *Conference on Empirical Assessment in Software Engineering*, 1999.

[Porter95] Porter, A., Votta, L. and Basili, V. R. "Comparing Detection Methods for Software Requirements Inspection: A Replicated Experiment" *IEEE Transactions on Software Engineering,* 21(6):563-575, 1995.

[Regnell99] Regnell, B., Runeson, P. and Thelin, T. "Are the Perspectives Really Different? - Further Experimentation on Scenario-Based Reading of Requirements", Technical

Report CODEN: LUTEDX(TETS-7172) / 1-40 / 1999 & local 4, Dept. of Communication Systems, Lund University, 1999.

[Runeson98] Runeson, P. and Wohlin, C. “An Experimental Evaluation of an Experience-Based Capture-Recapture Method in Software Code Inspections” *Empirical Software Engineering: An International Journal*, 3(4):381-406, 1998.

[Siegel88] Siegel, S. and Castellan, N. J. *Nonparametric Statistics for the Behavioral Sciences*, McGraw-Hill, Singapore, 1988.

[Thelin99] Thelin, T. and Runeson, R. “Robust Estimation of Fault Content with Capture-Recapture and Detection Profile Estimators” Accepted for publication at the *Conference on Empirical Assessment in Software Engineering*, 1999.

[Vander Wiel93] Vander Wiel, S. A. and Votta, L. G. “Assessing Software Design Using Capture-Recapture Methods” *IEEE Transactions on Software Engineering* 19(11):1045-1054, 1993.

[Wohlin95] Wohlin, C., Runeson, P. and Brantestam, J. “An Experimental Evaluation of Capture-Recapture in Software Inspections” *Software Testing, Verification and Reliability*, 5(4):213-232, 1995.

[Wohlin98] Wohlin, C. and Runeson, P. “Defect Content Estimation from Review Data” In *Proc. of the 20th International Conference on Software Engineering*. Pp. 400-409, 1998.

SESSION 4:

Novel Approaches in Software Process Assessments

The Role of the Client-Supplier Relationship in Achieving Software Quality

Jennifer Gasston
Software Quality Institute
School of Computing and Information Technology
Griffith University, Nathan Australia.

Abstract

This paper reports further on the findings from Gasston[1], with respect to human and organisational issues which impact the quality of software products. Within the context of this research project, the task of software development is seen to consist of a number of socially constructed processes enacted by human actors within an organisation. These processes are grouped into phases such as those identified in most software development life-cycle models. The paper explores the Software Requirements Phase of three projects within a medium-sized software development organisation in Australia. The study included assessment of both the processes used to produce the work products from the phase, and an evaluation of the quality of the work products themselves. Processes were assessed for both maturity and effectiveness. Software products were examined for the high-level quality attributes: accuracy, understandability, implementability, and adaptability [2].

The findings from the study suggest a mapping of human and organisational factors, relevant to process effectiveness [1], to the quality-carrying properties which contribute to the overall quality of the work products of the Software Requirements Phase. The study suggests that these effectiveness factors will have significant impact on software product quality. Addressing and managing the factors early in software development, during the requirements phase, may provide developers with the means of achieving the product quality goals of the project.

1. Introduction

Developers of software systems are primarily concerned with improving productivity, containing costs, particularly those associated with rework and maintenance, meeting schedules and improving the quality of the products produced so as to maintain or increase their position in a highly competitive industry. One approach to achieving these objectives is the implementation of a software process improvement program. Such programs are based on the premise that the routine and regular achievement of quality products requires a quality development process.

A top-down or bottom-up approach can be taken to process improvement. The top-down approach involves the comparison of an organisation's processes with generally accepted "best-practices" using a framework such as the CMM [3]or ISO15504 (SPICE) [4]. The aim is to implement changes to existing processes to bring them in line with "best practice" and in doing so, improve the product. On the other hand, the bottom-up approach relies on understanding the organisational context, processes, products, business objectives and strategy. Changes to processes should be guided by experience, not as in the top-down approach, by a set of standardised practices [5].

Davis[6] identifies the focus of achieving software quality through process maturity as a "quick fix" that is not sufficient in itself to achieve quality improvement. Evaluation of the usefulness of process improvement programs and other approaches to achieving quality software must be done within the context of the organisation. Baskerville et al [7] point out that "the underlying values of those proposing the methods [for example: The set of organisational norms or culture implied by prescriptive improvement frameworks and Quality Management Systems] and those of the analysts/developers applying the methods may be in conflict with the values, norms and ontology of the organisation". Too often the focus of an organisation is on achieving software process improvement without attempting to analyse the organisation's goals with respect to the product. Attempts must also be made to understand the context in which the development of the product takes place.

A number of human and organisational issues have been found to impact the effectiveness of software development processes [1,8]. Weinberg, Scacchi and

Demarco and Lister (cited in Curtis et al, [9]) argue that we must "understand how human and organisational factors affect the execution of software development tasks if we are to create software development technology that dramatically improves project outcomes"[9]. This paper suggests that, if we want to know what is really going on in an organisation with respect to the company's capability to produce high quality software products, software improvement programs must focus not only on the process of development, but also the development context and the products produced. Therefore, to explore the potential impact of software processes and the development context on the quality of the product, a case study was conducted in October 1995 in Australia.

The case organisation was ITCOSOUTH and two projects were examined PROJECTA and PROJECTB. PROJECTA was initiated from a client-produced specification document; whereas, PROJECTB, which was developed a year later for the same client, began from a client brief and was at the stage of systems testing at the time of the study. PROJECTA was examined historically as the project had been completed three months prior to the site visits. All project staff and documentation were accessible.

The tool, used in this study, is a framework for analysis, the Software Process/Product Quality Model (SPPQM)[1,8]. The model aims to identify: 1) potential areas for improvement; 2) organisational and human issues which impact on the effectiveness of the processes operating within a software development organisation; and 3) the extent to which software products exhibit desirable quality attributes has been used in this study. The framework involves:

- an evaluation of the effectiveness of the processes in terms of the organisational context (see Section 2.1);
- the assessment of the processes for process maturity (see Section 2.2);
- the examination of work products from the process in terms of their achievement of high- level quality attributes using the instrument (see Section 2.3).

Although the study involved the examination of a large number of processes within various software development phases, this paper focuses on the Software Requirements Specification process. Dromey [2] has identified the following desirable attributes of Software Requirements Specifications.: accurate, understandable, implementable and adaptable. Evaluating the accuracy of requirements alone, will involve an examination of the specifications for precision, correctness, completeness, consistency and traceability. Difficulties occur in identifying quality-carrying properties such as these which relate to the higher level quality attributes, making the assessment of the quality of various work products from the phases of software development difficult.

The paper argues that the issues associated with the client/supplier relationship within the case organisation had significant impact of the quality of the Software Requirements Specifications produced from the two projects. The paper discusses how the organisation, unable to harness the lessons learned on the first project, repeated the same mistakes on the second. It is suggested that it is essential that organisations identify and understand the human and organisational issues, which impact upon process effectiveness and which have the potential to significantly effect the quality of the products produced during the software development process.

2. Research Method

It was necessary, in this study, to examine software development processes both structurally (both from the perspective of the organisation and the organisational unit in which they manifest) and as a social process to be shaped, designed and used by human actors. If we are to increase our knowledge to the extent to which it can be successfully applied in practice or contribute in some way to improving software development within organisations, then it is necessary to ensure that the strategy taken to carry out the study provides us with a rich picture of software requirements specification processes in practice. Qualitative research allows us to "describe and illuminate the context and conditions under which research is conducted [10]" . A case study [11, p 54] can provide the organisational context for the study of the relationship between the software development process and the meeting of strategic business needs, which is an important factor under consideration when attempting to assess the effectiveness of that process. Therefore the case study methodology [11, pp99-102], in which the SPPQM has been used as a tool to both guide the collection and interpretation of data, has been used in this study.

2.1 Evaluating Software Process Effectiveness

Figure 1 [1], shows the indicators used in this study to evaluate software process effectiveness. Multiple data collection methods were used to explore each indicator. The use of focus group interviews [11,p335], both structured and unstructured interviews with individuals, and documentation examination was aimed at corroborating the same phenomenon, software process effectiveness, addressing a broader range of historical, attitudinal and behavioural issues, developing converging lines of inquiry. The focus group interview is a highly efficient qualitative data-collection technique. The objective is to obtain high-quality data in a social context where people can consider their own views in the context of the views of others[11, p335]. During the structured interviews with project team members who held different roles (eg. project directors, project managers, systems engineers, software engineers, quality assurance staff, testing officers), a questionnaire, based on the effectiveness factors identified in the SPPQM (Figure 1) was used to uncover specific issues. Interviewees outside the project team included the Quality Manager, Business Analysts, Account Executives, Strategic Planners, and the Information Systems Development Manager**.**

Various documents were examined to corroborate and augment evidence, gathered during interviews and direct observation, of performance of the processes under examination. Documents included, but were not limited to: Planning documentation, High Level and Low Level Software Specifications and Designs, and Review Records. Where documentary evidence proved contradictory, further inquiry was made into the specific issues and processes examined. This was done through focus and structured interviews.

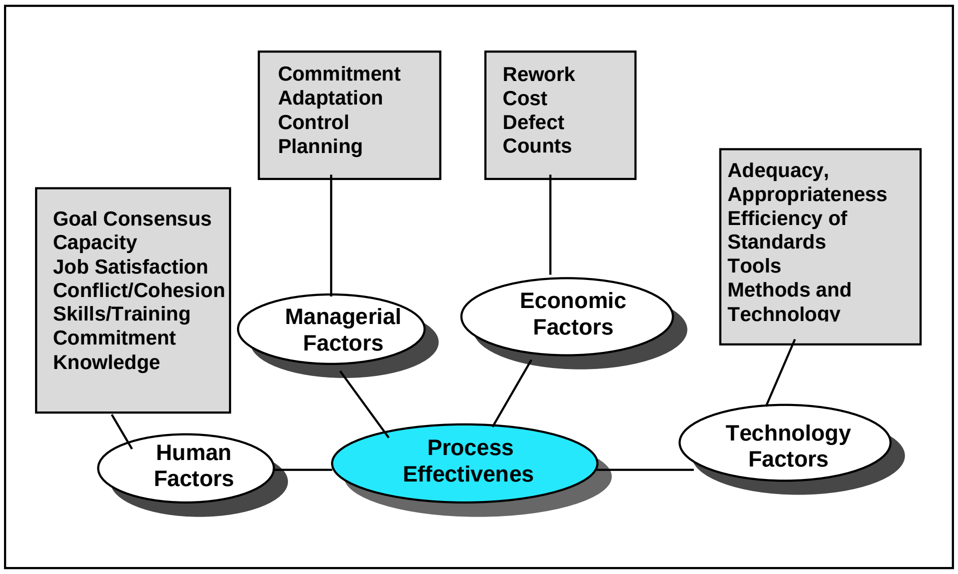


Figure 1: Process Effectiveness Indicators [1]

2.2 The Software Product Profile

A framework for constructing software quality models has been suggested by Dromey [2]. Suggestions are made as to how we might develop appropriate models for the requirements, design and implementation processes in software development. This framework has been used to develop the requirements product quality model used in the case study reported in this paper. The framework attempts to link tangible product characteristics to high-level intangible quality attributes. Four high-level attributes of software requirements are examined: accuracy, understandability, implementability, and adaptability [2]. The next step in constructing the model is to identify a set of quality-carrying characteristics which relate to the achievement of the high level attributes in order to assess whether products of the requirements processes meet desirable quality goals (Figure 2).

A set of questions, relating to the achievement of each characteristic, has been used during the evaluation process. The number of instances of each quality characteristic has been identified and recorded for each specification. The validity of the approach has been tested through the examination of the work products of the Requirements Analysis and Specification process within

ITCOSOUTH using Version 1 of the questionnaire. The quality-carrying properties listed under each attribute are those that have been found useful, at this stage of the project, in reflecting the high level attribute.

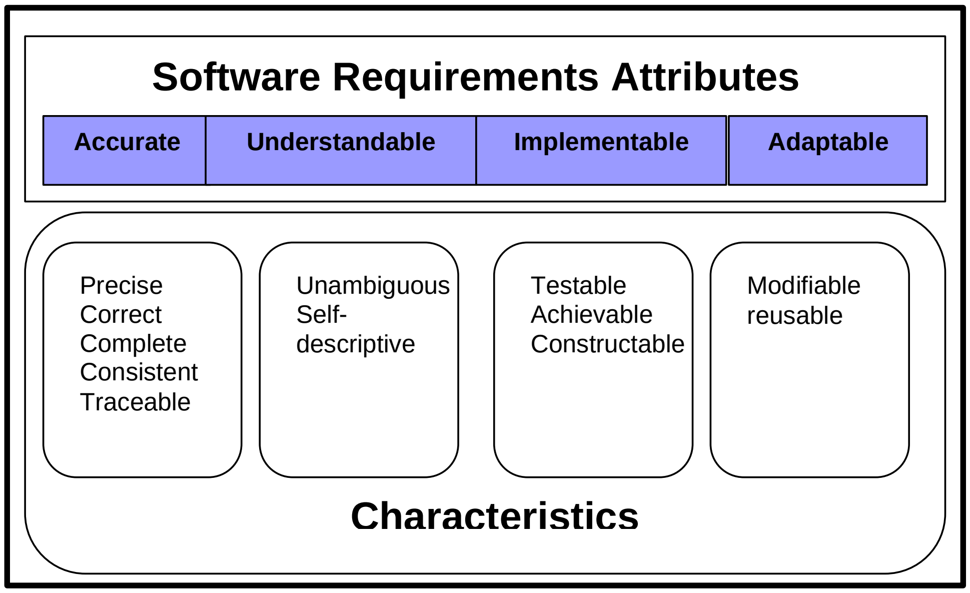


Figure 2 Software Requirements Product Profile Attributes

2.3 Evaluating Process Maturity

A Software Process Assessment model was used to evaluate the maturity of the processes under study. The Assessment model used was Part 5 Version 1.06 of the SPICE Document Set (ISO 15504)[4], the embedded model within the International Standard for Software Process Assessment. A Software Process Assessment Plan was produced for each assessment undertaken. Prior to visiting the research site to conduct the process assessments, a Process Performance Questionnaire was provided to the Quality Manager for the organisation and analysed to identify relevant interview and documentation review requirements. Each assessment was carried out by an assessment team led by the author. Checklists and unstructured interview questionnaires, as suggested in the SPICE guidelines for assessment, were utilised during the assessment process. Data gathered was analysed in terms of identifying process strengths and weaknesses and opportunities for improvement. Preliminary results were presented to the project personnel in ITCOSOUTH on the final day of the assessment. Feedback was obtained during this session and after presentation of the final assessment report to management two weeks after the assessment.

3. Findings

3.1 Process Maturity

The ISO15504 Part 5 framework was found useful in assessing the maturity of the processes examined. In order to satisfactorily assess the Requirements Specification process within ITCOSOUTH, one process was selected from the Engineering Process Category of the ISO15504 Framework, ENG 2 – Develop Software Requirements, and two from the Customer Supplier Process Category, CUS 3 – Identify Customer Needs and CUS 4 Perform Joint Audits and Reviews. It must be remembered in interpreting the results of the Software Process Assessment, that the version of the ISO15504 Part 5 framework for assessment used in 1995 to carry out the assessment has undergone a number of reviews and as a consequence is considerably different in structure to the current version.

The purpose of conducting the assessments, from the company's perspective, was to support the following overall goals:

1. Self Assessment of the software development processes at ITCOSOUTH
2. Determining the organisation's external capability.
3. Establishing best practices to guide the organisation in a process improvement program.
4. Establishing the company's position with regard to bench-marking information available through the SPICE Trials.

The key findings from the process assessments across both projects were:

- Globally, there was a lack of clear quality goals, particularly from a product perspective;

- There were particular weaknesses found in:

  - CUS3 - Identify customer needs where no training of systems development staff in domain knowledge was undertaken. Plans were found not to cover risk mitigation strategies, contingency plans, adequate scheduling of joint customer reviews and critical path analysis.

  - CUS4 - Perform Joint Audits and Reviews were there was evidence to suggest that the organisation was not really serious about joint reviews with the customer. Results of reviews were not monitored, so that the outcomes were not used for ongoing management. Most reviews consisted on individual document review rather than intensive group review processes.

The results of the assessment of ENG 2indicated a solid performance through Level 2 Significant strengths were evident in the allocation, definition and tracking of resource usage, particularly human resources At Level 3 capability, progress was being made towards a establishing a clearly defined process for developing software requirements. Generic Practice 3.1.1, Standardise the Process, was not rated as Fully Adequate as the standard process within the Quality Management System did not provide for base-lining of requirements. At Level 4, there was no evidence of the establishment of measurable quality goals for the work products of the process.

3.2 Evaluating Process Effectiveness

3.2.1 Organisational structure

From the evaluation of the effectiveness of the processes, the client-supplier relationship was found to be critical to the success of the processes of elicitation, development and specification of Software Requirements in the two projects examined in ITCOSOUTH. Both projects were developed for the same client organisation an internal division of the parent company. From the examination of meeting minutes, email messages and data collected during interviews with employees of ITCOSOUTH, who took part in the development of both systems, it was evident that a very strained relationship existed between the client's IT department and the supplier organisation. This appears to be primarily due to what the Quality Manager described as "historical baggage inherited at the formation of ITCOSOUTH". I must point out here that my analysis is limited to an examination of the development organisation only. A richer picture of events and perceptions may have been obtained had I been able to access the client organisation during this study.

It is the belief of the employees of ITCOSOUTH that the client organisation harbours a high degree of resentment towards them. The resentment, and some even go as far as to say antagonism, originates from the decision made by the parent company to take the IT/IS function out of individual divisions and to create an IT organisation servicing all divisions and subject to all the business requirements of any other division within the group.

Some IT staff within the client organisation transferred to ITCOSOUTH, but have since left mostly due to management conflict. Five personnel, who are responsible for conducting systems planning, systems identification and the management of development work performed by ITCOSOUTH, remain within the client company. Employees of ITCOSOUTH believe, based on numerous conversations, that it is the client's belief that the company charges too much for systems development and that the same systems can be obtained from other sources at cheaper prices or that the client organisation's IT department is capable of doing a better job themselves.

The Project Director advised that ITCOSOUTH encourages its clients to seek quotations from outside organisations as the company is confident that they are providing high quality services at a very reasonable cost. In some cases, he suggested, the company would be very pleased if a "difficult" client, such as the one discussed now, would seek services elsewhere.

3.2.2 Domain Knowledge

A distinct lack of domain knowledge was found within ITCOSOUTH regarding client organisations, particularly with respect to internal divisions. No attempt had been made to harness the knowledge of original employees by transferring lessons learned across projects to organisational memory. There was no evidence that management considered domain knowledge important to the customer-supplier relationship, since its acquisition was not part of the company's induction procedure. The project manager for PROJECTB, a contractor, stated that he considered his lack of domain knowledge with respect to the client's systems, had contributed greatly to his problems in establishing an effective client-supplier relationship on that project. There was evidence that email messages requesting clarification of requirements issues were ignored by the client representative for the project.

Staff of ITCOSOUTH try to avoid being assigned to development projects for the client. Often contract staff are put on the projects, which appears to exacerbate the situation, since they lack domain knowledge and are often unaware of the obstacles that may be placed in their way. The issues associated with the client-supplier relationship on both projects were found to have a significant effect on staff job satisfaction and morale.

There was consensus between the project managers that the main "antagonist" within the client's IT/IS staff is a former employee of ITCOSOUTH. The contractor, assigned as project manager to PROJECTB, encountered considerable problems in obtaining user requirements for the project. In many cases it was suggested to him "… you should know what we need. You should know all about our systems." In general, the project managers suggested that "the client is unprofessional and dealing with them is a nightmare. It is as if they are saying to us "I'm the customer and you'll do what I tell you. They were not interested in listening to our ideas." When PROJECTB's manager tried to gain access to client organisation to try to acquire some domain knowledge, barriers were put up not only by the client but also by ITCOSOUTH. Pressure was put on him, as a contractor, to perform and he suggested that his employer felt that "wasting time at the client's premises would be too costly." Though there was evidence from management meetings suggesting that management's perception is that "this particular client is entirely unreasonable', there appeared to have been little upper management involvement or support on both projects.

It appeared, form the point of view of the project team, that the client representative's goal was to demonstrate a lack of capability on the part of ITCOSOUTH. There is a potential for goal disagreements to engender conflict and political activity in organisations [12, p 69]. It is generally agreed that power characterises relationships among social actors (an individual, sub-unit or organisation) [12, p3]. The distribution of power within an organisation can also become legitimated over time, so that those within the setting expect and value a certain pattern of influence.

By adopting the role of single client representative, the “antagonist” has used his position to control project team access to users and information vital to the success of the requirements elicitation process. “In most cases, positions which are given to powerful social actors as a consequence of their power also provide those actors with additional power due to the information and decisions that are within the purview of these positions” [12, p57]. By ignoring the situation, management of ITCOSOUTH legitimated the power of the client representative into authority. “…to negotiate the treacherous waters of computing demands from different managerial and user groups, a developer needs political skills as well as ones of systems analysis.” [13]. The implication is then, that management sought to ignore the problem rather than address it by focusing on the acquisition of domain knowledge and the development of an effective relationship with the client.

3.2.3 Lack of User Involvement

The project team on PROJECTA were in intermittent contact with the client representative who was a person from the IT/IS department of the client organisation. It is a policy of ITCOSOUTH that the client organisation provide users to conduct acceptance testing. It was not until this request was conveyed to the client that the project team became aware that the development of the system had not been known to the potential users. As a consequence it was not until that time that substantial problems were identified with system functionality and interfaces to existing systems. Six major changes to the scope of the project became necessary just prior to the proposed release of the system.

Since the majority of the system analysis was performed by the client, it appears that their own lack of user involvement may have been the main contributor to the scope problems encountered both during the time of production of a functional design, and in the later stages of the project implementation.

ITCOSOUTH's project team should have also ensured that adequate user representation was obtained from the client organisation. Client management involvement, outside the client's IT department, may have been necessary to secure an adequate level of co-operation

3.2.4 Lack of Goal Consensus

The project managers and team leaders stated that the way in which job goals are set up within the organisation causes difficulties. There is no time for full induction processes. Employees are allocated time to learn new techniques and skills, whereas learning the customer's business is not seen as important. This is largely due to the perception on the part of management, and many employees themselves, that domain knowledge is not a marketable skill. There is a three month induction program for new graduates which includes domain knowledge, but no provision is made for new "experienced" employees or contractors. With the increase in outsourcing and contract appointments in the IT industry, appropriate and adequate induction processes, which focus on the acquisition of client business knowledge, appears to be an essential requirement both now and in the future. If information systems are to be utilised for competitive advantage it is essential that analysts have the knowledge to identify information systems development requirements that will strategically benefit the organisation.

3.2.5 Lack of Strategic Focus

At the time of this study, the adoption of a long-term strategic view to systems development for client organisations had only just emerged within ITCOSOUTH. Prior to that time, applications were, on the whole, developed independently; there was no real integration of functions across client organisations through the use of IT.

The services of ITCOSOUTH's Strategic Planning Services Group were not sought by the client on these two projects. The planning group has evidence, from client satisfaction surveys conducted by the group, to suggest that a higher level of system acceptance appears to be achieved from projects involving client organisations who rely heavily on the services of the planning group. During the planning process the following issues are analysed:

- Understanding the current situation;
- Business directions;
- What the client is hoping to achieve;
- The critical success factors or specific objective;
- The current systems; and
- Process modelling, business functions.

It appears that many of the problems incurred during the later stages of PROJECTA were a result of an incomplete, often ambiguous set of user requirements produced by the client organisation. The initial requirements document produced by the client was used by the project team as the preliminary specification. The ambiguity and lack of completeness can be linked to non-achievement of the high-level quality attributes accuracy and understandability. The project team's interpretation of the system was not understandable to the client and vice versa, and proved to be useless for effective communication of requirements between client and supplier.

3.2.6 Limited Exposure to Client's Business Needs

The specialist database group within ITCOSOUTH takes on almost a limited liaison role between the client and the project team. The group's role encompasses data architecture, database design, and the provision of data analysis and design skills and services to project teams. Although the manager of the group suggested that his people have a broader view of the client's business with respect to data requirements and system interfaces, he admitted that they have limited exposure to the actual client organisation. Discussions are limited to certain requirements where they are able to detect impacts of changes based on their understanding of the client's complete data model. The group has experienced similar difficulties in dealing with the client's IT/IS department as "they believe they know what they want and as a result it is harder for the database group to gain acceptance." The group manager suggested ".. the way they [the client's IS/IT Department] operate, a lot of requirements don't surface until the database group actually do the work … lots of scope changes because the scope wasn't broad enough. The original budget may only be 50% of what is required." The database group see the project team as their customer, and the project manager confirmed the existence of a formal internal relationship between the project and the design groups.

Unless a strong working relationship can be achieved between suppliers and customers, difficulties such as those experienced by ITCOSOUTH and their client will continue to occur. Particularly this is important right from the onset of a project, due to the impact that an ineffective requirements analysis process can have on the whole project. No attempt was made at that time, on the part of management to address, either at the operational nor strategic levels, the necessity to ensure client-supplier co-operation throughout the lifetime of projects.

"The expertise of the IS professional could lie in knowing where to look for information, knowing how to screen it and interpret it, and knowing how to utilise it internally for specific purposes to create knowledge and expertise within their own organisation." [14]. Organisations must be aware that individual members will acquire, disseminate and interpret information from various sources of situated knowledge within the organisation and as a result lead to behavioural change and performance improvement It is important therefore that organisations identify these potential sources, so as to obtain optimal benefits from software process assessment and improvement programs [15].

"The importance of situated knowledge, of local experience and a grounded understanding of priorities within particular application areas, confirms that software development is still a highly interpretative process, at least for large numbers of on-site workers (Fincham et al, 1994 cited in Beirne et al [13]). From a series of case studies Beirne et al [13] found that most effective developers were often considered to be those who could network informally with users, and lay claim to organisational and commercial knowledge as well as technical expertise.

PROCESS MATURITY	PROCESS EFFECTIVENESS	PRODUCT PROFILE
• Weakness in Training program – induction processes. • Lack of Risk mitigation strategies, contingency plans. • Insufficient joint customer reviews scheduled • Customer reviews not monitored nor outcomes monitored	***Management Factors*** • No focus on organisational learning • Communication problems – lack of client confidence • Customer satisfaction not seen as important ***Historical issues*** organisational structure ***Commitment:*** • Lack of leadership from management – ignored situation, not proactive; • Lack of commitment on part of project team. • Lack of strategic focus Job satisfaction, Staff Morale: • Staff avoided projects for client; • Motivation, Communication problems: • Work not acknowledged. Lack of ***goal consensus:*** • Client/Project teams. • Management/Project teams. Conflict: Project teams/Client	***Understandability*** Requirements often ambiguous. Ambiguity of terms. ***Accuracy:*** Terms used inconsistently; incomplete requirements, not well defined; requirements missing. Lack of traceability to source documentation. ***Implementable:*** Not all requirements testable; ***Adaptable*** Difficulties in modifying requirements found due to documenting problems.

Table1: Issues shown to impact on the quality of the Requirements Specification Documents

Table 1 summarises the findings of the study identifying the key weaknesses within the process, the issues which contributed to ineffective processes and the resultant evaluation of the attributes of the products, the Requirements Specification documents.

5. Conclusions

The findings of this study suggest a mapping of human and organisational factors, relevant to process effectiveness, to the quality-carrying properties which contribute to the overall quality of the work products from the Software Requirements Phase. The study also suggests that the effectiveness of software development processes will have significant impact on software product quality. The particular issues within the client-supplier relationship found in this study to effect product quality are:

- Organisational structure;
- Domain knowledge of the software development team;
- Lack of user involvement;
- Political agendas;
- Lack of goal consensus;
- Lack of strategic focus;
- Limited exposure to client's business needs.

These issues in turn, impacted staff morale, conflict/cohesion within the project team and between the customer and the team, and management. The problems identified in the company's relationship with the client were apparently clear to management during the first project, but by choosing to do nothing, management exacerbated the situation.

The client has knowledge about business goals and organisational needs, and unless project teams attempt to identify this information during the requirements elicitation process, and attempt to link social system objectives to the technical objectives, then implementation and system acceptance problems will occur. Communication breakdown, and the political agendas that have taken place within ITCOSOUTH have been found to hinder the elicitation process. By identifying and addressing potential factors within the organisation that may effect the quality of work products, software developers may be able to increase their chances of developing high quality software products.

References

1. Gasston, J. (1996) “Process improvement: an alternative to BPR for software development organisations”, Software Quality Journal, 5 pp171-183. Chapman & Hall UK.

2. Dromey, R.G. “Cornering the Chimera”, IEEE Software,Vol 13, No. 1, Jan. 1996

3. Paulk, M.C.; Curtis, B.; Chrissis, M.B.; Weber, C.V.. (1993) CMU/SEI-93-TR-24 Capability Maturity Model for Software, Version 1.1, SEI, Pittsburgh, PA., February 1993.

4. ISO-SPICE (Software Process Improvement and Capability dEtermination),Special initiative for Software Process Assessment Standardisation, ISO/IEC JTC1/SC7/WG10, 1993-96.

5. McGarry, F. (1994) “Top-Down vs. Bottom-Up Process Improvement”, IEEE Software Vol 11, No 4, July1994. Pp 12-13.

6. Davis (1993) “Software Lemmingineering”, IEEE Software, 1993, 10, 5, pp 79-84.

7. Baskerville, R., Travis J. and Truex, D. (1992) “Systems without method: the impact of new technologies on information systems development projects”, in The Impact of Computer Supported Technologies on Information Systems Development, Kendall, K.E. et al (eds) Elsevier Science Publishers, North-Holland, IFIP 1992.

8. Gasston, J. and Rout, T. (1994) “Can the effectiveness of software processes be assessed?” Software Quality Journal, 3 pp153-166, Chapman & Hall UK.

9. Curtis, B., Krasner H., and Iscoe, (1988) “A field study of the Software Design Process for Large Systems”, Communications of the ACM, November 1988, Volume 31, Number 11.

10. Kaplan, B. and Duchon, D. (1988) “Combining Qualitative and Quantitative Methods in Information Systems Research: A Case Study”, MIS Quarterly, 12, PP571-588.

11. Patton, M. Q. (1990) Qualitative Evaluation and Research Methods, Second Edition, SAGE Publications, Inc. Newbury Park, CA. USA.

12. Pfeffer, J, "Power in Organisations", Pitman Publishing Inc., Massachusetts,USA, (1981).

13. Beirne, M. Ramsay, H. Panteli, A. for P. Thompson and C. Warhust (eds), (1998) Workplaces of the Future, Macmillan Publishers, London. UK . "Close encounters of the Nerd Kind: Control and contradiction in the computing labour process".

14. Swan, J.A. & Galliers, R.D.(1996) "Networking: The Future of Information Systems" The DATA BASE for Advances in Information Systems , Fall 1996 (Vol 27, No.4)

15. Gasston, J. and Halloran, P. (1999) "Continuous Software Improvement Requires Organisational Learning", forthcoming in Proceedings of the 7th International Conference on Software Quality Management, Southampton, UK 29/3/99 to 1/4/99.

Improving Market-Driven RE Processes

Pete Sawyer, Ian Sommerville and Gerald Kotonya
Lancaster University
Lancaster, UK

Abstract

Orthodox requirements engineering process models have evolved to support the needs of organisations developing bespoke software. These models adapt poorly to the specification and development of market-driven software products. This is unfortunate because market-driven software developers are economically important and face quite distinct problems. Time-to-market is typically the overriding constraint of market-driven products and, for many organisations, this squeezes the resources available for performing the requirements process and for its improvement. In this paper we examine the background to these problems and propose pragmatic and lightweight measures that can help organisations adapt orthodox requirements engineering good practice to their market-driven businesses.

1. Introduction

Market-driven software poses particular problems for how requirements are elicited and handled. While many of these problems have been addressed by the requirements engineering community, they have been addressed outside the context of market-driven product development. Several authors have examined the problems of market-driven software [Lubars 93, Potts 95, Kamsties 98, Deifel 98]. Our contribution is to synthesise a set of good practice guidelines from their work and to place these within the context of our Requirements Engineering Good Practice Guide (REGPG) [Sommerville 97, Sawyer 97].

Market-driven software comes in many forms and targets many different kinds of market. Many of the problems that we discuss seem to be particularly acute for small-to-medium enterprises (SMEs) [Kamsties 98] selling to niche markets.

We suspect that larger companies face similar problems but their size and resources can distort the picture. For the purposes of this paper we are concerned with the following categories of software product:

- End-user products. Examples include CASE tools, Internet browsers and mail tools. Time-to-market is the overriding constraint on these systems' development projects. If timely release is achieved, the generic properties they must exhibit in order to consolidate their market include usability and conformance to industry standards (e.g. look-and-feel, data formats, communication protocols).

- Componentware (sometimes called commercial-off-the-shelf software - COTS) intended to be integrated with other components to comprise end-user applications. These include real-time operating systems, signal processing algorithms, object request brokers and database management systems. Time-to-market and conformance to standards are also crucial for these products. Usability is an issue only in respect to the APIs and language bindings that they offer the application programmer.

Some products don't fit neatly into either category and many products can be used stand-alone or as componentware. However, both categories of what we will henceforth refer to as *packaged* software pose a number of problems that are distinct from those facing bespoke software. The most crucial of these is that a packaged software product that is late risks permitting a more timely competitor to shape the market by absorbing latent demand and establishing a de-facto standard. For market-driven products, time-to-market is not merely a constraint but a "survival attribute" [Novorita 96]. Nor does time-to-market only constrain the initial release. Once the product's market share has been established, it needs to be retained or expanded. This typically requires timely incremental releases designed to increase functionality, track market and technological trends, and rectify defects.

The rigid constraint of time-to-market restricts the available options when projects fall behind schedule. Adding resources is often impractical, especially for an SME. The preferred solution to schedule slip is normally to concentrate resources on meeting the most critical requirements and so release the product on time with the core functions intact, but with fewer cosmetic features than originally planned.

For timely release to be effective, there must be user demand for the product and orthodox requirements engineering practice stresses the need to elicit users' requirements and use these as the basis for subsequent development. However, at product conception, there will be only *potential* customers and it is unlikely that these will share the product "vision". Hence, for packaged software, "requirements are invented, not elicited" [Lubars 93].

Once the release has been achieved real users will exist and new requirements will start to emerge. An implication of this is a need for a dynamic process of elicitation, evaluation and prioritisation which must dovetail closely with an iterative release cycle. Unfortunately, most requirements practices have evolved to support bespoke software development and a more linear development process model. Many market-driven software developers consequently experience difficulty adapting these to their environment.

Many orthodox requirements practices act as an overhead in the short term and so conflict with time-to-market. However, product longevity is also important so the benefits of (e.g.) requirements management are of important in the long-term. There is strong evidence [Kamsties 98] that many packaged software developers waste resources compensating for the long-term deficiencies in the quality of their requirements processes.

2. Requirements engineering process maturity

Good requirements engineering practice is applied only patchily throughout the software industry [El Eman 95]. This seems to be particularly true in the packaged software sector; not because of any inherent failing by the companies developing packaged software, but because the particular demands of the sector have not been the focus of the requirements engineering community.

Widening the use of good practice is how the software process improvement (SPI) [Zahran 98] movement aims to stimulate overall improvements in quality. Unfortunately, few SPI models or quality standards have much to say on requirements engineering. In the Capability Maturity Model (CMM) [Paulk 93], for example, the single explicitly requirements engineering -related *key practice area* (KPA) is that of requirements management. This mandates that requirements are allocated, that requirements changes are reviewed and that resources and responsibilities are assigned accordingly. One of the effects of this has been to stimulate awareness of the importance of requirements management.

Unfortunately, implementing requirements management is a major stumbling block [Fowler 98] if organisations have insufficiently robust requirement processes in place to ensure that requirements are allocated correctly, or that requirements changes are identified and analysed. The CMM offers no help on these underpinning measures and SPI programmes are often capped by weaknesses in organisations' requirements processes. "the CMM definition gave scant attention to the issue of whether the requirements to be managed are the 'right' requirements" [Hutchings 95].

In general terms, requirements processes are less well understood, less well supported by standards and less mature than other software processes. This motivated the REAIMS project to develop the REGPG to extend the principles of SPI into the requirements process. The REGPG advocates incremental improvement based upon the phased adoption of established good practice. It recognises that different companies will have different maturity levels for requirements engineering and uses a three-level maturity model similar to the lower three levels of the CMM as shown in Figure 1 (see [Sawyer 97] for a fuller discussion of the motivation for this architecture).

Initial level organizations have an *ad hoc* requirements process. They find it hard to estimate and control costs as requirements have to be reworked and customers report poor satisfaction. The processes are not supported by planning and review procedures or documentation standards. They are dependent on the skills and experience of the individuals who enact the process.

Repeatable level organizations have defined standards for requirements documents and have introduced policies and procedures for requirements management. They may use tools and methods. Their documents are more likely to be of a consistent high quality and to be produced on schedule.

Defined level organizations have a defined process model based on good practices and defined methods. They have an active process improvement programme in place and can make objective assessments of the value of new methods and techniques.

This exploits the CMM's approach to goal-driven improvement and makes the principle familiar to the many organisations already embarked upon wider SPI programmes. The latter point is important because recognition of weaknesses in requirements processes is often the result of undertaking an SPI programme.

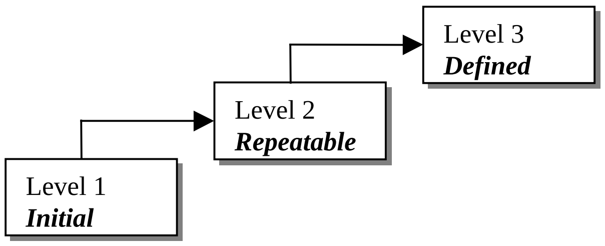


Fig. 1. The 3-level REAIMS process maturity model

The REGPG is organised as a set of 66 good practices that address each of the process areas listed in the next section. In recognition of the fact that good practices vary in their ease of adoption and the support measures they need to be effective, the good practices are rated according to whether they are *basic*, *intermediate* or *advanced*. Basic practices represent fundamental measures which underpin a repeatable process. In this paper, we concentrate exclusively on identifying basic practices for packaged software development.

In its present form, the REGPG provides generic guidelines for requirements engineering process improvement. In the sections below, we suggest how some of these should be interpreted by packaged software developers and suggest a number of packaged software-specific practices. Our ambitions are modest, however. We concentrate on basic practices that can help develop a repeatable process since the evidence is that the requirements processes of many packaged software developers, particularly SMEs, are at the initial level.

3. Requirements processes for packaged software

Time-to-market means that when problems occur, resources, functionality and quality are squeezed, typically requiring a reduction in planned functionality. The challenge is to ensure that the available resources are concentrated on meeting the most cost-effective requirements. Effective prioritisation and cost/impact assessment are therefore key issues to help "make acceptable tradeoffs among sometimes conflicting goals such as quality cost and time-to-market; and allocate resources based on the requirement's importance to the project as a whole" [Karlsson 97]. How they are performed and integrated with the requirements process is critical to the success of packaged software.

The REPEAT process [Regnell 98] is a successful example of how this can done by focusing on the incremental acquisition, analysis and selection of requirements for each release. Unfortunately, processes like REPEAT that impose control on requirements handling and provide defences against unforeseen project difficulties are rare for packaged software. It is normal for organisations to have an ad-hoc process with little explicit guidance for choosing the decision-making personnel, selecting the decision-making procedures to be used or the recording of those decisions.

With ad-hoc processes, time pressure will tend to make the selection of which requirements to implement less rational and may cause short cuts to be taken. Examples might include skipped reviews, documents not being maintained, etc. These may result in reduced quality of the product and/or of ancillary products such as documents. [Kamsties 98] and [Lubars 93] highlight the fact that delivery deadlines naturally lead to the longer-term benefits of many requirements practices being traded off against short-term exigencies. For example, while documenting and maintaining the customer requirements is a cornerstone of most orthodox requirements processes, many packaged software producers fail to do either [Kamsties 98]. This is often because: there is no contractual requirement for such a document; document-based requirements validation is seen as neither appropriate nor effective; the company may have evolved from a small start-up company with a clear product vision and has never evolved a document-based culture; and maintenance of requirements documents is an overhead on an already stretched workforce.

We think that some of these are valid reasons for not producing a *physical* requirements document. However, failure to maintain the requirements in some persistent, retrievable and traceable form inevitably causes problems in the medium and long term. For example, [Kamsties 98] describes companies which fail to document their product's requirements, allowing freedom to quickly develop the product according to the product vision. However, the problem of developing test cases stimulates the retrospective development of the requirements. This risks wasting effort verifying the product, and commonly leads to the discovery of new requirements that conflict with the implementation. The need for expensive rework of tacit requirements is therefore common. Early identification and documentation of the requirements would enable conflicts to be identified and traded off far more inexpensively.

Perhaps the most obviously distinctive feature of market-driven projects is that the product is conceived by the developer (perhaps a product visionary) rather than commissioned by a customer. The orthodox view of the requirement process is that one first understands the problem, then specifies the solution. With packaged software, identification of the problem may follow identification of the solution, which in turn may spring from identification of an opportunity offered by a combination of market conditions and technology advances. Thus, to paraphrase two possible scenarios for product conception, either:

- the marketing department says "market studies suggest there's a market for a product that does X, could we build one?"; or

- the technical arm of the organisation says "technology now allows us to do X, is there a market for a product that does X?".

It is an article of faith in requirements engineering that requirements have to be elicited from customers and, in particular, from those who will use the system. It is recognised that elicitation is difficult and error-prone and a number of techniques have evolved to help the process. Unfortunately, in the absence of a commissioning customer, few of these techniques which allow direct developer-user contact are directly useful. [Keil 95] shows that, at product conception, marketing is by far the most common source of user requirements for packaged software. Techniques such as QFD [Haag 96] can be used to help analyse and weight the requirements but they cannot fully compensate for the absence of direct user-developer contact.

This poses significant risk for the developer since they cannot be certain that they have addressed users' requirements until relatively late in the process. Prototypes may be used to help validate the invented requirements and beta versions may be released to favoured customers. However, the whole process of developing a requirements baseline is very much less stable than for a well-managed bespoke software project. In contrast to bespoke software, the development organisation is initially the primary stakeholder in the product and undertakes its development to meet their strategic business objectives. To be successful, these need to be consistent with the requirements of their customers.

Once the product is released, real customers and users will exist and these should be tapped as a source of requirement for subsequent releases of the product. Technical support staff can be a crucial conduit for customer requirements [Keil 95] but the quality of the requirements information from technical support is crucially dependent on their training and motivation. Once an effective conduit for users' requirements has been identified, a mechanism has to be provided to allow the requirements to be handled.

The crucial point is that the developer's association with their product should be a long-term one. It is likely that far more effort will go into maintaining the product through incremental releases than initial delivery. This contrasts strongly with bespoke software where maintenance seldom colours the developer's requirements process even though maintenance frequently costs the customer more than delivery. While recognising the need to get the original concept right, the packaged software requirements process must be tailored to the needs of long-term, incremental development in the face of evolving market conditions and technological opportunities.

Below, we list the main distinctive features and requirements of a market-driven requirements process. These are listed according to the activities and process products used to organise the practices described in the Requirements Engineering Good Practice Guide (REGPG) [Sommerville 97, Sawyer 97].

The Requirements Document: This is not a contractual requirement. There may be no company culture of documenting requirements.

Requirements Elicitation: The developer is the primary stakeholder. Developer requirements are derived from strategic business goals and market opportunities. At system concept, there are only potential users/customers. User requirements are normally elicited indirectly via market studies. Developer and user requirements have to be balanced to derive the core requirements. Once in service, users/customers will generate requirements for new versions/releases. Some user/customer requirements will be in the form of bug reports. As the company evolves or moves into new markets, it may lose up-to-date domain knowledge that regular contact with bespoke system clients can give.

Requirements Analysis and Negotiation: Users/customers and the developer may have very diverse requirements. Priority and cost must be balanced. Satisfaction of some requirements may be deferred to a later version/release.

Describing Requirements: Requirements documents are not normally read by customers.

System modelling: Conceptual models of customers' business environments will be speculative.

Requirements Validation: The requirements baseline will evolve for each release. Customer/user validation may take place only once substantial implementation investment has been made (e.g. at a trade fair)

Requirements Management: There may be no company culture of managing requirements. The effort needed to maintain the product in the market will exceed the effort needed to enter the market. Time-to-market constraints impose a strong tension between between short-term expediency and the long-term benefits of managing requirements. The requirements baseline may remain fluid until late in the development cycle. Requirements processes will be enacted concurrently with down-stream development activities.

Not all the characteristics listed above are unique to packaged software. For example, many bespoke software developers have no company culture of managing requirements. However, we believe that the characteristics are more prevalent in packaged software. Our conclusion from this is that these are the things that must be tackled in order to begin to effect an overall improvement in handling requirements for packaged software.

A few development organisations, like those described in [Regnell 98] and [Hutchings 95] are already well advanced with their packaged software requirements process improvement programmes. Many other organisations will deny that they have a problem. The empirical evidence from other sectors of the software industry is that at least some of these organisations will find that problems do emerge over time. Many more organisations are becoming aware of problems as their products evolve, as their company size increases, as they are forced to respond to market pressures or as they seek to adopt standards (e.g. ISO9001-3).

Our core belief is that most organisations, especially those that have experienced rapid growth from small beginnings, find it hard to adapt orthodox practices to their requirements processes and have too few resources to devote to meta-level activities such as process improvement. The challenge facing the requirements engineering and SPI communities is to provide guidance on pragmatic measures to begin a process of incremental improvement using trusted good practice with clear benefits for the adopter.

4. Requirements practices for packaged software

This section lists the good practices that we think address each of the process areas listed above. We do not claim that the practices are sufficient to deliver a repeatable requirements process but we believe that they all necessary. The list represents our interpretation of the observations and recommendations of [Deifel 98, Hutchings 95, Kamsties 98, Keil 95, Lubars 93, Novorita 96, Potts 95]. In several cases we have directly adopted these authors' recommendations.

Practices in italics are generic to all types of software and have been taken directly from the REGPG. Note that not all of these are classified as basic practices in the REGPG. However, in the special context of packaged software, we believe that they are all basic. It does not necessarily follow from this that their cost of introduction and application is high. However, some of the practices will be relatively expensive. Unfortunately, space does not allow a detailed discussion of the practices' probable cost.

The Requirements Document

- At the product concept stage, document the business goals and user requirements
- For each release, baseline the high-level requirements at a fixed cut-off time
- *Define a Standard Document Structure*
- *Make the Document easy to change*

Requirements Elicitation

- Define procedures for receiving, analysing and documenting requirements derived from users' in-service experience
- Train and motivate technical support staff
- *Identify and consult system stakeholders*
- *Use business concerns to drive requirements elicitation*

Requirements analysis and negotiation

- For new releases, evaluate the cost of meeting each requirement
- For each release, review all the requirements including those that weren't selected for the last release
- *Prioritise requirements*

Describing requirements

- *Use standard templates for describing requirements*
- *Supplement natural language with other descriptions of requirements*

System modelling

- *Develop complementary system models*
- *Model the system architecture*

Requirements validation

- *Organise formal requirements inspections for each release*
- *Use multi-disciplinary teams to review requirements*
- *Define validation checklists*
- *Use prototyping to animate requirements*
- *Propose requirements test cases*

Requirements management

- *Uniquely identify each requirement*
- *Define policies for requirements management*
- *Define change management policies*
- *Use a tool to manage requirements*
- *Record rejected requirements*

5. Summary and conclusions

In this paper we have synthesised a number of good practices for requirements engineering for packaged software. We have exploited the results from a number of recent papers that have examined the differences between packaged software and bespoke software. These have drawn a number of conclusions about the characteristics of requirements engineering for packaged software and, in some cases, identified good practice. We have summarised the lessons and integrated their packaged-software good practices with our own REGPG, explaining how the REGPG's existing good practices should be interpreted for packaged software.

We believe that many of the problems experienced by packaged software developers have their root in fundamental failings in their requirements processes so we have concentrated on basic practices that, we believe, pay the highest dividends. Most of these are organisational or procedural practices; very few are technical in nature. Some of these may be quite expensive to adopt, some may require a fundamental shift in company culture, and most will only start to pay off in the medium to long term. However, we justify this by observing that most packaged software developers aim to develop and maintain products with a long shelf life. This means that they bear the direct cost of maintaining their products' market viability so their relationship with their products and their customers is necessarily a long-term one.

There are other classes of software and software development models that, while not specifically product-oriented, share many of the characteristics of the packaged software to which we addressed this paper. These include Rapid Application Development (RAD) where the emphasis is on tightly time-constrained development cycles. The DSDM [Stapleton 97] consortium has defined a RAD process model that advocates the substitution of close user/developer relations for much of the detailed documentation mandated by orthodox development models. This appears to be a close fit to the needs of many packaged software developers except that as we have described, close user/developer relations are hard to achieve for packaged software developers at system concept. Once the product is on the market and has real users, however, it is possible that approaches like DSDM may prove more readily adaptable for packaged software developers.

6. References

[Deifel 98] Deifel, B.: "Requirements Engineering for Complex COTS", Proc. Fourth International Workshop on Requirements Engineering: Foundations for Software Quality (REFSQ'98), Pisa, Italy, 1998.

[El Eman 95] El Eman, K., Madhavji, N.: "Measuring the Success of Requirements Engineering Processes" Proc. 2nd IEEE International Sympoium on Requirements Engineering (RE93), York, UK, 1995.

[Fowler 98] Fowler P., Patrick, M., Carleton, A., Merrin, B.: "Transition Packages: An Experiment in Expediting the Introduction of Requirements Management", Proc. Third International Conference on Requirements Engineering (ICRE'98), Colorado Springs, Co., 1998.

[Haag 96] Haag, S., Raja, M., Schkade, L.: "Quality Function Deployment Usage in Software Development", Communications of the ACM, 39 (1), 1996.

[Hutchings 95] Hutchings, A., Knox, S.: "Creating Products Customers Demand", Communications of the ACM. 38 (5), 1995.

[Kamsties 98] Kamsties, E., Hörmann, K., Schlich, M.: "Requirements Engineering in Small and Medium Enterprises: State-of-the-Practice, Problems, Solutions and Technology Transfer", Proc. Conference on European Industrial Requirements Engineering (CEIRE'98), Hammersmith, UK, 1998.

[Karlsson 97] Karlsson, J., Ryan, K.: "A Cost-Value Approach for Prioritizing Requirements", IEEE Software, 14 (5), 1997.

[Keil 95] Keil, M., Carmel, E.: "Customer-Developer Links in Software Development", ", Communications of the ACM. 38 (5), 1995.

[Lubars 93] Lubars, M., Potts, C., Richter, C.: "A Review of the State of the Practice in Requirements Modelling", Proc. IEEE International Sympoium on Requirements Engineering (RE93), San Diego, Ca., 1993.

[Novorita 96] Novorita, R., Grube, G.: "Benefits of Structured Requirements Methods for Market-Based Enterprises", Proc. International Council on Systems Engineering (INCOSE) Sixth Annual International Symposium "Systems Engineering: Practices and Tools", July 7 - 11, 1996 Boston, Massachusetts.

[Paulk 93] Paulk, M., Curtis, W., Chrissis, M., Weber, C.: Capability Maturity Model for Software, Version 1.1, CMU/SEI-93-TR-24, Software Engineering Institute, USA, 1993.

[Potts 95] Potts, C.: "Invented Requirements and Imagined Customers: Requirements Engineering for Off-the-Shelf Software", Proc. 2nd IEEE International Sympoium on Requirements Engineering (RE93), York, UK, 1995.

[Regnell 98] Regnell, B., Beremark, P., Eklundh, O.: "Requirements Engineering for Packaged Software - A Baseline Process in a Situated Improvement Programme", Proc. Conference on European Industrial Requirements Engineering (CEIRE'98), Hammersmith, UK, 1998.

[Sawyer 97] Sawyer, P., Sommerville, I., Viller, S.: "Requirements Process Improvement Through the Phased Introduction of Good Practice, Software Process Improvement and Practice, **3**, (1) 1997.

[Sommerville 97] Sommerville, I., Sawyer, P.: Requirements Engineering - A Good Practice Guide, John Wiley, 1997.

[Stapleton 97] Stapleton, J.: DSDM Dynamic Systems Development Method: The Method in Practice, Addison-Wesley, 1997.

[Zahran 98] Zahran S.: Software Process Improvement Practical Guidelines for Business Success, Addison-Wesley, 1998.

Conformance Analysis of the Tailored CMM with ISO/IEC 15504

Yingxu Wang, Alec Dorling, Judith Brodman* and Donna Johnson*
Centre for Software Engineering, IVF
Argongatan 30, S-431 53, Molndal, Sweden
Tel: +46 31 706 6174, Fax: +46 31 27 6130
Email: {Yingxu.Wang, Alec.Dorling}@ivf.se
*LOGOS International, Inc.
8 Mackintosh LN, Lincoln, MA 01773, USA
Email: {brodman, johnson}@tiac.net

Abstract

This paper reports a case study on compliance analysis between software process models. Conformance of the Logos tailored CMM (T-CMM) with the ISO/IEC 15504 process reference model is analysed. The method and coverage of this work are based on the ISO/IEC 15504 requirements defined in the reference model. Frameworks of the T-CMM model and ISO/IEC 15504 process reference model are briefly described. Mappings between T-CMM and ISO15504 are carried out in two-directions. Compliant level between T-CMM and ISO/IEC 15504 is analysed based on comparison and contrast of their frameworks and mappings between the two models. This paper intends to develop a practical approach to process models' compliance analysis, and to provide a systematic perspective and a number of interesting findings on the features of the T-CMM and ISO/IEC 15504 models.

Keywords: Software engineering, software process, process assessment and improvement, tailored CMM, ISO/IEC 15504, mapping, conformance analysis

1. Introduction

It is strongly expected to systematically and comparatively analyse the current software engineering process standards and models in software engineering research and in the software industry [1-6]. This paper reports a case study in analysing the conformance of a CMM-derived process model, the Logos tailored CMM (T-CMM) [7-8], with ISO/IEC 15504 [9-12]. This work is based on the conformance requirements defined in [10] that cover model purpose, scope, elements and indicators, mapping, and capability translation.

To enable the conformance analysis, frameworks of the T-CMM model and ISO/IEC 15504 process and capability models are described and contrasted in Sections 2 and 3 respectively. Then, in Section 4, mappings between T-CMM and ISO/IEC 15504 are carried out mutually. Compliant level of T-CMM to ISO/IEC 15504 is analysed based on comparison of their frameworks and process rating methods, and quantitative mappings between the process models in Section 5.

2. The T-CMM model

T-CMM [7-8] is a tailored SEI CMM [13-15] for small business, organisation and projects conducted by Logos International in 1995. T-CMM was derived based on the work in [16-17] and has been recognised by SEI and the sponsor of the tailoring project [8].

The motivation for a tailored CMM was based on the Logos experience with nearly 200 small organisations for process improvement. The Logos discovered that small organisations and projects were encountering difficulties applying the CMM to their software process improvement effort, because the CMM largely reflects the practices of large software organisations [16-17]. As a result, software organisations that are small or have small projects were experiencing vast difficulty in implementing process improvement programs based on the CMM and, thus, have not progressed very high on the software process maturity scale.

The tailoring has been focused on improving SEI CMM usability on: a) documentation overload; b) layered management: c) scope of reviews overkill; d) limited resources; e) high training costs; and f) unrelated practices. The strategy of the tailoring was to produce a tailored CMM that maintained the intent, structure and key process areas (KPAs) of the CMM [13-15], and will be suitable to small organisations. Therefore only the CMM key practices (KPs) were tailored and revised to make them applicable for small organisations. The tailoring has been carried out by: a) clarification of existing practices; b) exaggeration of the obvious; c) introduction of alternative practices; and d) alignment of practices with small organisations and projects' structure and resources. Despite the fact that 82% of the CMM KPs were modified in the tailoring process, the changes that were introduced into the CMM did not radically change its structure, especially at the KPA level.

This section describes the structure of T-CMM framework. The process model and capability model of T-CMM are summarised in this section.

2.1 The T-CMM process model

The process taxonomy of the T-CMM is described by terms of system,

Table 1. The structure of KPAs in T-CMM

CL	KPA	Description
CL_1		Initial
CL_2		Repeated
	KPA_{21}	Requirements management
	KPA_{22}	Software project planning
	KPA_{23}	Software project tracking and oversight
	KPA_{24}	Software subcontract management
	KPA_{25}	Software quality assurance

	KPA_{26}	Software configuration management
CL_3		Defined
	KPA_{31}	Organisation process focus
	KPA_{32}	Organisation process definition
	KPA_{33}	Training programme
	KPA_{34}	Integrated software management
	KPA_{35}	Software product engineering
	KPA_{36}	Intergroup coordination
	KPA_{37}	Peer reviews
CL_4		Managed
	KPA_{41}	Quantitative process management
	KPA_{42}	Software quality management
CL_5		Optimised
	KPA_{51}	Defect prevention
	KPA_{52}	Technology change management
	KPA_{53}	Process change management

capability levels (CLs), key process areas (KPAs), common features, and key practices (KPs). T-CMM identified a set of 18 KPAs and 150 KPs within five capability levels [7-8]. A hierarchical categorisation of the KPAs in T-CMM is shown in Table 1.

2.2 The T-CMM capability model

In the capability dimension of T-CMM, the process capability is defined at five levels as shown in Table 2. Each capability level of T-CMM is featured by a number of KPAs with defined KPs as shown in Table 2, except at level 1 there is no defined KPAs since this level is treated as the baseline for an initial software organisation in T-CMM. The capability maturity levels and the related KPAs in assessment are predefined and fixed according to T-CMM model.

Table 2. Capability levels of T-CMM

CL	Description	KPAs	KPs
CL_1	Initial	0	0
CL_2	Repeated	6	62
CL_3	Defined	7	50
CL_4	Managed	2	12
CL_5	Optimised	3	26

3. The ISO/IEC 15504 model

This section describes the ISO/IEC 15504 framework based on the ISO/IEC 15504 process reference model [10]. The process model and capability model of ISO/IEC 15504 are summarised in this section.

3.1 ISO/IEC 15504 process model

A hierarchical structure of the ISO/IEC 15504 processes is shown in Table 3, where the LC, PC and PR stand for life cycle, process category and process respectively.

Table 3. Hierarchical structure of ISO/IEC 15504 processes

LC	PC	PR	Sub. PR	Description
Primary				Primary life cycle processes
	CUS			Customer-supplier
		CUS.1		Acquisition
			CUS.1.1	Acquisition preparation
			CUS.1.2	Supplier selection
			CUS.1.3	Supplier monitoring
			CUS.1.4	Customer acceptance
		CUS.2		Supply
		CUS.3		Requirements elicitation
		CUS.4		Operation
			CUS.4.1	Operational use
			CUS.4.2	Customer support

	ENG			Engineering
		ENG.1		Development
			ENG.1.1	System requirements analysis and design
			ENG.1.2	Software requirements analysis
			ENG.1.3	Software design
			ENG.1.4	Software construction
			ENG.1.5	Software integration
			ENG.1.6	Software testing
			ENG.1.7	System integration and testing
		ENG.2		System and software maintenance
Supporting				Supporting life cycle processes
	SUP			Support
		SUP.1		Documentation
		SUP.2		Configuration management
		SUP.3		Quality assurance
		SUP.4		Verification
		SUP.5		Validation
		SUP.6		Joint review
		SUP.7		Audit
		SUP.8		Problem resolution

		SUP.9		Measurement
		SUP.10		Reuse
Organisational				Organisational life cycle processes
	MAN			Management
		MAN.1		Management
			MAN.1.1	Project management
		MAN.2		Quality management
		MAN.3		Risk management
	ORG			Organisation
		ORG.1		Organisational alignment
		ORG.2		Improvement process
			ORG.2.1	Process establishment
			ORG.2.2	Process assessment
			ORG.2.3	Process improvement
		ORG.3		Human resource management
		ORG.4		Infrastructure
Total	5	23	17	

3.2 ISO/IEC 15504 capability model

In the capability dimension of ISO/IEC 15504, process capability is defined at six levels and with nine intermediate process capability attributes as shown in Table 4. These capability levels generally incorporate two process attributes (sub-levels) except the level *1* (one attribute) and level *0* (no attribute).

Table 4. The ISO/IEC 15504 capability rating scale

Capability level	**Process attribute**	**Description**
	$PA_{5.2}$	Continuous improvement
	$PA_{5.1}$	Process change
CL_5		Optimising process
	$PA_{4.2}$	Process control
	$PA_{4.1}$	Process measurement
CL_4		Predictable process
	$PA_{3.2}$	Process resource
	$PA_{3.1}$	Process definition
CL_3		Established process
	$PA_{2.2}$	Work product management
	$PA_{2.1}$	Performance management
CL_2		Managed process
	$PA_{1.1}$	Process performance
CL_1		Performed process
CL_0		Incomplete process

4. Mutual mapping between T-CMM and ISO 15504

In this section, correlation between T-CMM and ISO/IEC 15504 is analysed in both directions. Mutual mapping between T-CMM and ISO/IEC 15504 is carried out.

4.1 Correlation between T-CMM and ISO/IEC 15504

According to the analysis in [1-3, 6], mapping between a pair of models is insymmetric. Therefore correlation analysis and mutual mapping between T-CMM and ISO/IEC 15504 are carried out in two directions as shown in Fig.1.

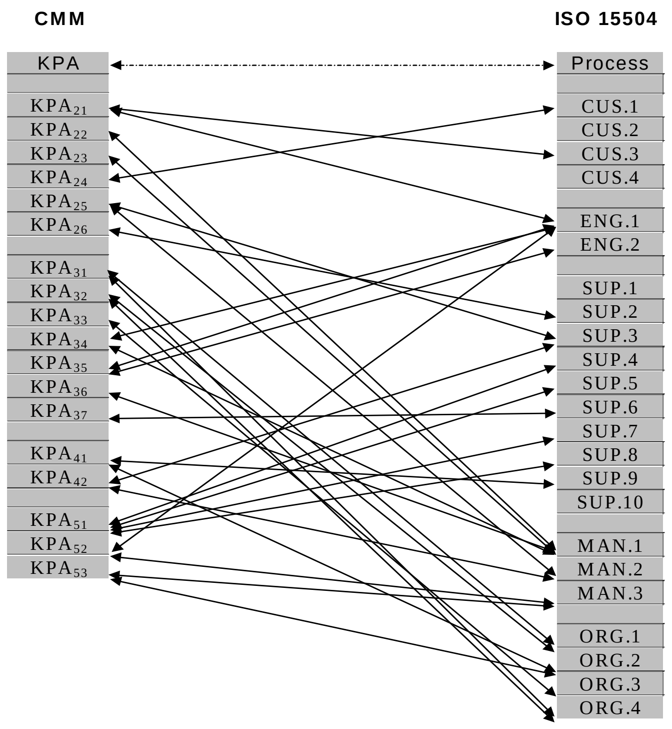


Fig. 1 Mapping between T-CMM and ISO/IEC 15504

In Fig.1, the left column lists the 18 KPAs of T-CMM; the right column lists the 23 processes of ISO/IEC 15504. The lines with two-directional arrows show the correlation or equivalency between T-CMM KPAs and ISO/IEC 15504 processes.

Fig.1 shows there are one-to-one, one-to-many and many-to-one correlation between the two models, since different organisation of process structures in T-CMM and ISO/IEC 15504. In Fig.1, it can be found that all T-CMM KPAs are covered by the ISO/IEC 15504 processes. The fact that all T-CMM KPAs can be fully mapped onto ISO/IEC 15504 is the foundation that would enable T-CMM conforming to ISO/IEC 15504. Although, reversibly, not all ISO/IEC 15504 processes are matched in T-CMM.

4.2 Mapping T-CMM onto ISO/IEC 15504

Based on the correlation net developed in Fig.1, T-CMM can be mapped onto ISO/IEC 15504 at the KPA level [5,18]. A detailed mapping of T-CMM onto ISO/IEC 15504 is shown in Table 5.

For characterising degrees of correlation between the two models, four confident levels of correlation between processes are modelled in terms *very high* (V), *high* (H), *low* (L) and *not* (N) correlation. This approach is designed to refine the measurement of the degrees of correlation between processes, rather than to provide a simple conclusion as ‘yes’ or ‘no’ as adopted in conventional methods [18]. In Table 5, a letter in the squared brackets shows the confident level of correlation in a mapping.

Table 5. Mapping T-CMM onto ISO/IEC 15504

CL	KPA	Description	Correlated process(es) in ISO 15504 [with a confident level]
CL_1		Initial	
CL_2		Repeated	
	KPA_{21}	Requirements management	CUS.3 [H], ENG.1 [L]
	KPA_{22}	Software project planning	MAN.1 [L]
	KPA_{23}	Software project tracking & oversight	MAN.1 [L]
	KPA_{24}	Software subcontract management	CUS.1 [H]
	KPA_{25}	Software quality assurance	SUP.3 [V], MAN.2 [V]
	KPA_{26}	Software configuration management	SUP.2 [V]
CL_3		Defined	
	KPA_{31}	Organisation process focus	ORG.1 [H], ORG.4 [L]
	KPA_{32}	Organisation process definition	ORG.1 [V], ORG.4 [H]
	KPA_{33}	Training programme	ORG.3 [H]
	KPA_{34}	Integrated software management	ENG.1 [L], MAN.1 [H]

	KPA_{35}	Software product engineering	ENG.1 [H], ENG.2 [L]
	KPA_{36}	Intergroup coordination	MAN.1 [L]
	KPA_{37}	Peer reviews	SUP.6 [V]
CL_4		Managed	
	KPA_{41}	Quantitative process management	SUP.9 [H], ORG.2 [L]
	KPA_{42}	Software quality management	SUP.3 [H], MAN.2 [V]
CL_5		Optimised	
	KPA_{51}	Defect prevention	SUP.4 [H], SUP.5 [L], SUP.7 [L], SUP.8 [L]
	KPA_{52}	Technology change management	ENG.1 [L], MAN.3 [L]
	KPA_{53}	Process change management	MAN.3 [L], ORG.2 [H]

4.3 Mapping ISO/IEC 15504 onto T-CMM

Based on the correlation net shown in Fig.1, ISO/IEC 15504 can also be mapped onto T-CMM at the process level [1,2,5]. A detailed mapping of ISO/IEC 15504 onto T-CMM is shown in Table 6. A letter in the squared brackets shows the confident level of correlation in mapping as defined in Subsection 4.2.

Table 6. Mapping ISO/IEC 15504 onto T-CMM

LC	**PC**	PR	**Description**	**Correlated KPA(s) in CMM [with a confident level]**
Primary			Primary life cycle processes	
	CUS		Customer-supplier	
		CUS.1	Acquisition	KPA_{24} [H]
		CUS.2	Supply	
		CUS.3	Requirements elicitation	KPA_{21} [H]
		CUS.4	Operation	
	ENG		Engineering	
		ENG.1	Development	KPA_{21} [L], KPA_{34} [L], KPA_{35} [H], KPA_{52} [L],
		ENG.2	System & software maintenance	KPA_{35} [L]
Suppor -ting			Supporting life cycle processes	
	SUP		Support	
		SUP.1	Documentation	
		SUP.2	Configuration management	KPA_{26} [V]
		SUP.3	Quality assurance	KPA_{25} [V], KPA_{42} [H]
		SUP.4	Verification	KPA_{51} [H]
		SUP.5	Validation	KPA_{51} [L]
		SUP.6	Joint review	KPA_{37} [V]
		SUP.7	Audit	KPA_{51} [L]
		SUP.8	Problem resolution	KPA_{51} [L]
		SUP.9	Measurement	KPA_{41} [H]
		SUP.10	Reuse	

Organi-sational			Organisational life cycle processes	
	MAN		Management	
		MAN.1	Management	KPA_{22} [L], KPA_{23} [L], KPA_{34} [H], KPA_{36} [L]
		MAN.2	Quality management	KPA_{25} [V], KPA_{42} [V]
		MAN.3	Risk management	KPA_{52} [L], KPA_{53} [L]
	ORG		Organisation	
		ORG.1	Organisational alignment	KPA_{31} [H], KPA_{32} [V]
		ORG.2	Improvement process	KPA_{41} [L], KPA_{53} [H]
		ORG.3	Human resource management	KPA_{33} [H]
		ORG.4	Infrastructure	KPA_{31} [L], KPA_{32} [H]

5. Conformance analysis of T-CMM with ISO 15504

Based on the mutual mapping as described in Section 4, the levels of correlation and conformance between T-CMM and ISO/IEC 15504 are quantitatively analysed in this section.

5.1 Correlation level in mapping T-CMM onto ISO/IEC 15504

For quantitatively analysis the complicated correlation between two process models in forms one-to-one, many-to-one and one-to-many, a set of numeric weights is introduced. The confident levels of correlation as shown in Tables 5 and 6 are weighted as: V := 10, H := 7, L := 3, and N := 0. Based on this, the correlation levels of the 18 KPAs in T-CMM with regard to the ISO/IEC 15504 processes can be calculated as shown in Fig.2. The numeric weighting approach has also enabled the development of mapping algorithms and tools between process models [19].

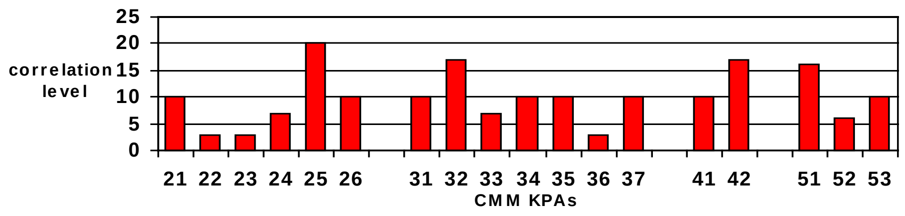


Fig. 2 Correlation level in mapping T-CMM onto ISO/IEC 15504

In Fig.2, the highest correlative KPAs are KPA_{25} - software quality assurance, KPA_{32} - organisation process definition, KPA_{42} - software quality management, followed by KPA_{51} - defect prevention; The lowest correlative KPAs are KPA_{22} - software project planning, KPA_{23} - software project tracking and oversight, and KPA_{36} - intergroup coordination. There is no KPA in T-CMM that can not be mapped onto the ISO/IEC 15504 processes. This fact indicates there is a basis of compliance between T-CMM and ISO/IEC 15504 according to the requirements in [10].

5.2 Correlation level in mapping ISO/IEC 15504 onto T-CMM

In the same approach as developed in Subsection 5.1, the correlation levels of the 23 ISO/IEC 15504 processes with the T-CMM KPAs are quantitatively derived as shown in Fig.3. In Fig.3, the highest correlative processes are MAN.2

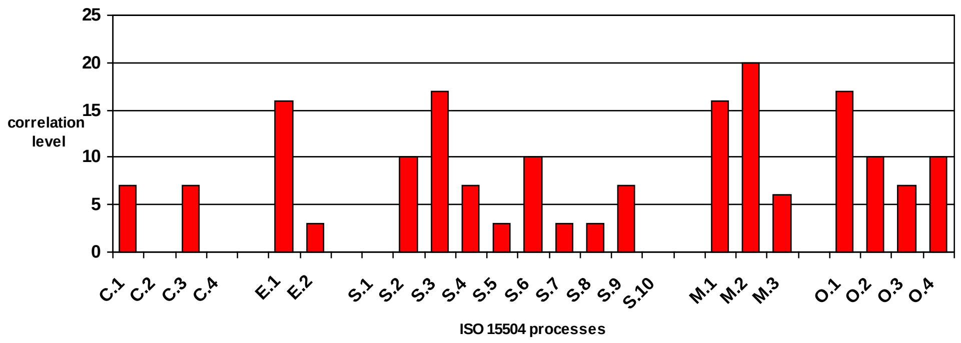


Fig.3 Correlation level in mapping ISO/IEC 15504 onto T-CMM

- quality management, SUP.3 - quality assurance and ORG.1 - organisational alignment, followed by ENG.1 - development and MAN.1 - management.

It is noteworthy that, as shown in Fig.3, there are four processes in ISO/IEC 15504, such as CUS.2 - supply, CUS.4 - operation, SUP.1 - documentation, and SUP.10 - reuse, without correlation to the T-CMM KPAs. However, this fact does not affect the compliance level that T-CMM maps onto ISO/IEC 15504, because a compliant mapping is defined in the reversed way as in [10].

5.3 Conformance between T-CMM and ISO/IEC 15504

Generally the compliant relationship between T-CMM and ISO 15504 is summarised in Table 7, where the process models, capability models, capability rating methods and rating results of T-CMM and ISO/IEC 15504 are comparatively contrasted.

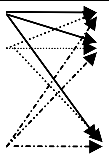
Table 7 shows that:

a) The process models of T-CMM and ISO/IEC 15504 are highly correlative,

   especially at the KPA/process level. Although the fact that T-CMM KPAs are assigned into separate capability levels is an exception, which contradicts to the requirement of that "every process should can be evaluated in any capability levels [10]" according to ISO/IEC 15504.

b) The capability models between T-CMM and ISO/IEC 15504 are also highly correlative. Minor difference in the capability models is that ISO/IEC 15504 has 9 generic attributes for all processes; while T-CMM has 5 common features for grouped KPAs.

Table 7. Degree of conformance between T-CMM and ISO/IEC 15504

Aspect	**CMM**	**ISO/IEC 15504**	**Confident level in correlation**
Process model			
	Grouped in 5 capability levels	Modelled in 5 process categories	L
	18 KPAs	23 Processes	V
	150 KPs	201BPs	H
Capability model			
	5 levels	6 levels	V
	5 common features	9 attributes	H
Rating method			
	{Φ} ➨ CL_1 {KPA_{21} - KPA_{26}} ➨ CL_2 {KPA_{31} - KPA_{37}} ➨ CL_3 {KPA_{41} - KPA_{42}} ➨ CL_4 {KPA_{51} - KPA_{53}} ➨ CL_5	PR_1 CL_0 PR_2 CL_1 ....  PR_{23} CL_5	H
Rating result			
	Capability level of a project or organisation	Capability profile of processes	H
Summary			Average correlation level: 7.29,

			ie. V > Average-level > H

Notes: Confident level in mapping: V - very high equivalent, H - high equivalent, L - low equivalent,

N - non equivalent

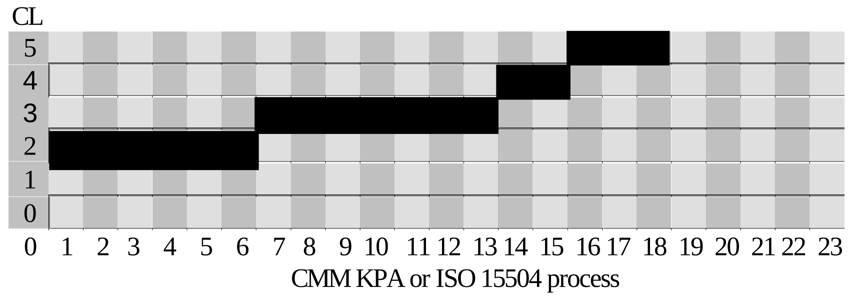


Fig. 4. Domain of capability levels between T-CMM and ISO/IEC 15504

c) The rating method of T-CMM is a subset of ISO/IEC 15504 as shown in Table 7 and Fig.4. In Fig.4, the T-CMM capability areas of the 18 KPAs are marked by black colour. As a subset of ISO/IEC 15504 capability domain, T-CMM rating method is conformance to ISO/IEC 15504.

d) The rating results of T-CMM and ISO/IEC 15504 have highly equivalent meaning. Only difference is that ISO/IEC 15504 results in a capability profile of a set of processes; while the T-CMM represents a single process capability level for a project or organisation. A supplement method for filling this gap has been developed in [4,6] that derives an aggregated capability level at project or organisation level from the ISO/IEC 15504 process capability profile.

This section is designed to provide a complete perspective on the compliance between the T-CMM and ISO/IEC 15504 models. As shown in Table 7, the average compliant level between T-CMM and ISO/IEC 15504 is 7.29 in the scale of 10, which indicates a confident conformance level between very high (V) and high (H) correlation.

6. Conclusions

Objective conformance analysis between existing models is an important and yet difficult research subject. This paper has analysed the compliant level between T-CMM and the ISO/IEC 15504 process reference model based on the conformance criteria as defined in [10]. The frameworks of the T-CMM model and ISO/IEC 15504 process and capability models have been described. Mappings between T-CMM and ISO/IEC 15504 have carried out in two-directions. Conformance of T-CMM with ISO/IEC 15504 has analysed based on the comparison and contrast of their process and capability models, as well as their rating methods and rating results. The general findings is that, although there are minor and historical differences, T-CMM is 72.9% compliant to ISO/IEC 15504 in the aspects of process and capability dimensions, and in their capability rating methods and results.

The work reported in this paper can be taken as a detailed case study of conformance analysis between an established process model and the ISO/IEC 15504 standard. Based on this work, recommendations have been provided to both T-CMM and ISO/IEC 15504 towards a completed compliance.

7. Acknowledgements

The authors would like to acknowledge many helpful discussions with colleagues of ISO/IEC JTC1/SC7/WG10. We would like to thank the valuable comments of the referees for improving the quality of this paper.

References

[1] Wang, Y., Court, I., Ross, M., Staples, G., King, G. and Dorling, A. (1997), Quantitative Analysis of Compatibility and Correlation of the Current SPA Models, *Proceedings of the IEEE International Symposium on Software Engineering Standards (ISESS'97),* California, USA, pp.36-56.

[2] Wang, Y., Court, I., Ross, M., Staples, G., King, G. and Dorling, A. (1997),Quantitative Evaluation of the SPICE, CMM, ISO 9000 and BOOTSTRAP, *Proceedings of the IEEE International Symposium on Software Engineering Standards (ISESS'97),* California USA, June, pp. 57-68.[3] Wang, Y., Court, I., Ross, M., Staples, G. and Dorling, A. (1997),Comparative Assessment of a Software Organisation with the CMM andSPICE, *Proceedings of the 5th International Conference on Software Quality Management (SQM'97),* March, Bath, UK, pp.S4.1-11.

[4] Wang, Y. (1999), How to Relate Process Capability Levels betweenCurrent Process Standards and Models, Lecture Notes Session IX – Process Capability, *Proceedings of European Winter School on Software Process,* EC ESPRIT Basic Research WG – PROMOTER II, January, France, pp.IX.1- IX.89.

[5] Wang, Y., Court, I., Ross, M., Staples, G. and King, G. (1996), Towards aSoftware Process Reference Model (SPRM*), Proceedings of InternationalConference on Software Process Improvement (SPI'96),* Brighton UK,pp.145-166.

[6] Wang Y., King, G., Doling, A. and Wickberg, H. (1999), A UnifiedFramework of the Software Engineering Process System Standards and Models, *Proceedings of 4th IEEE International Software Engineering Standards Symposium (ISESS'99),* IEEE CS Press, Brazil, May, pp.P1.1-10.

[7] Broadman, J. G. and Johnson, D. L (1995), *The LOGOS Tailored CMM[SM] for Small Businesses, Small Organisations and Small Projects (V.1.0),* LOGOS International Inc., August, USA, pp. 1-26.

[8] Johnson, D. L and Broadman, J. G. (1997), Tailoring the CMM for SmallBusinesses, Small Organisations, and Small Projects, *Software Process Newsletter*, No.8, pp. 1-6.

[9] ISO/IEC JTC1/SC7/WG10 (1998), TR 15504-1: Information Technology – Software Process Assessment - *Part 1: Concept and introduction guide (V. 3.3),* pp.1 - 11.

[10] ISO/IEC JTC1/SC7/WG10 (1998), TR 15504-2: Information Technology – Software Process Assessment - *Part 2: A Reference Model for Process and Process Capability (V. 3.3),* pp.1 - 39.

[11] ISO/IEC JTC1/SC7/WG10 (1998), TR 15504-4: Information Technology – Software Process Assessment - *Part 4: Guide to Performing Assessments, (V. 3.3),* pp.1 - 18.

[12] ISO/IEC JTC1/SC7/WG10 (1998), TR 15504-5: Information Technology – Software Process Assessment - *Part 5: An Assessment Model and Indicator Guidance (V. 3.3),* pp.1 - 121.

[13] Humphrey, W.S. and W.L. Sweet (1987), A Method for Assessing theSoftware Engineering Capability of Contractors, *Technical Report CMU/SEI-87-TR-23,* Software Engineering Institute, Pittsburgh, Pennsylvania, USA.

[14] Paulk, M.C., Curtis, B., Chrissis, M.B. and Weber, C.V. (1993),Capability Maturity Model for Software, Version 1.1, *Technical reportCMU/SEI-93-TR-24*, Software Engineering Institute, Pittsburgh, Pennsylvania, USA.

[15] Paulk, M.C., Weber, C.V., Garcia, S., Chrissis, M.B. and Bush, M. (1993),Key Practices of the Capacity Maturity Model, Version 1.1, *TechnicalRreport CMU/SEI-93-TR-25*, Software Engineering Institute, Pittsburgh,Pennsylvania, USA.

[16] Brodman J.G. and Johnson D.L. (1994), What Small Business and SmallOrganisation Say about the CMM, *Proceedings of the 16th International Conference on Software Engineering (ICSE16),* Sorrento, Italy, pp.331-340.

[17] Johnson, D. L and Broadman, J. G. (1992), Software Process Rigors YieldStress, Efficiency, *Signal Magazine,* August, USA.

[18] Paulk, M.C., Konrad, M.D. and Garcia, S.M. (1994), CMM Versus SPICE Architectures, *Software Process Newsletters,* Spring , pp.7-11.

[19] Dorling, A., Wang, Y., Kirchhoff, U., Sundmaeker, H., Maupetit, C., Pitette, G., Pereira, J. and Hansen, S. (1999), *ICT Acquisition Process Assessment Methodology*, The PULSE Consortium Publication, March, pp.1-87

SESSION 5:

Software Measurement

Empirical Studies of Inspection and Test Data

Reidar Conradi
NTNU,
Trondheim, Norway

Amarjit Singh Marjara
Gap Gemini AS,
Trondheim, Norway

Børge Skåtevik
STC,
Vatlandsvåg, Norway

Abstract

Inspections and testing represent core techniques to ensure reliable software. Inspections also seem to have a positive effect on predictability, total costs and delivery time.

This paper presents a case study of inspections and testing, done at the Ericsson development department outside Oslo in Norway. This department develops and maintains customer-defined services around AXE phone switches, i.e. the functionality around the “star”" and “square” buttons on house telephones.

AXE development at Ericsson worldwide uses a simple, local experience database to record inspections and testing data. Two MSc students from NTNU have been given access to such historical data in 1997 [Marjara97] and 1998 [Skaatevik99]. The results from these two diploma theses constitute the basis for this paper.

The paper will study questions such as:

– The effectiveness and cost-efficiency of inspections,

– The cost-effectiveness and defect profile of inspection meetings vs. individual inspections,

– The relation between complexity/modification-rate and defect density,

– Whether the defect density for modules can be predicted from initial inspections over later phases and deliveries.

The paper is organized as follows.

Section 1 summarizes some relevant parts of the state of the art, especially of inspections.

Section 2 first describes the Ericsson context, and

section 3 describes questions hypotheses for the study.

Section 4 describes the organization of the study, and

Section 5 presents and discusses the results.

Section 6 sums up the paper and recommends some future work.

1. Introduction

The paper will present results from two MSc theses at NTNU, that have analyzed historical defect data at Ericsson in Oslo, Norway -- related to their AXE switches. Ericsson has practiced Gilb inspections for many years, and collects defect data from inspections and testing in a small database.

These studies revealed that inspections indeed are the most cost-effective verification technique. Inspections tend to catch 2/3 of the defects before testing, by spending 10% of the development time and thereby saving about 20% of the time (by earlier defect correction, a ``win-win'').

Inspection meetings were also cost-effective over most testing techniques, so they should not be omitted. Inspection meetings also found the same type of errors (Major, Super Major) as individual inspections.

We also found that there is a correlation between module complexity, modification rate, and the defect density found during field use, but not during inspections and test.

Due to missing data, we could not find out whether the defect density of modules repeated itself across inspection/test phases and over several deliveries, i.e. we could not predict ‘‘defect-prone” modules.

However, defect classification was unsatisfactory, and prevented analysis of many interesting hypotheses.

2. State of the art

Quality in terms of reliability is of crucial importance for most software systems.

Common remedies are sound methods for system architecture and implementation, high-level languages, formal methods and analysis, and inspection and testing techniques. Especially the latter two have been extensively described in the literature, and vast empirical materials have been collected, analyzed and published.

This paper only refers to general test methods, so we will not comment on these here.

Inspections were systematized by Fagan [Fagan76] [Fagan86] and represent one of the most important quality assurance techniques. Inspections prescribe a simple and well-defined process, involving group work, and have a well-defined metrics. They normally produce a high success rate, i.e. by spending 10% of the development time, we diagnose 2/3 of the defects before testing, and save 20% of the total time -- a win- win: so quality is ‘‘free”. Inspections can be applied on most documents, even requirements [Basili96]. They also promote team learning, and provide general assessment of reviewed documents.

Of current research topics are:

- The role of the final inspection meeting (emphasized by Tom Gilb [Gilb93], see also [Votta93].
- When to stop testing?, cf. [Adams84].
- The effect of root-cause-analysis on defects.
- The role of inspection vs. testing in finding defects, e.g. their relative effectiveness and cost.
- The relation between general document properties and defects.
- Defect densities of individual modules through phases and deliveries.

Our research questions and hypotheses deal with the three latter.

3. The company context

Ericsson employs about 100,000 people worldwide, whereof 20,000 in development. They have company-wide and standardized processes for most kind of software development, with adaptations for the kind of work being done.

Ericsson has adopted a classical waterfall model, with so-called "tollgates" at critical decision points. In all this, verification techniques like inspections and testing are crucial. Inspection is done for every life-cycle document, although we will mostly look at design and code artifacts. Testing consists of unit test, function test and system test, where the two latter may be done at some integration site different from the development site (e.g. Stockholm).

We will only study results from design inspections, simplified code reviews and partly testing in this paper.

The inspection process at Ericsson is based on techniques originally developed by Michael Fagan [Fagan76] at IBM and refined by Tom Gilb [Gilb93]. The process is tailor-made by the local development department. In addition there is a simplified code review done by individual developers (data from code review and unit test are sometimes merged into a “desk check"). Thus full inspections are only done upon design documents in the context of this paper.

Data from inspections/reviews and testing are collected in a simple, proprietary database and used for local tuning of the process. Defects are classified in Major, SuperMajor and Questions (the latter is omitted later) -- thus no deep categorization.

We have studied software development at the Ericsson site outside Oslo. It just passed CMM level 2 certification in Oct. 1998, and aims for level 3 in year 2000.

The Oslo development site has about 400 developers, mostly working on software. The actual department has about 50 developers, and works mostly on the AXE-10 digital software switch, which contains many subsystems. Each subsystem may contain a number of modules. The development technology is SDL design language (SDT tool from Telelogic) and their proprietary PLEX language from the late 1970s (own compilers and debuggers).

Special inspection groups are formed, called product committees (PC), to take care of all impacts on one subsystem. In this paper, we will only look at subsystem-internal inspections, not across subsystems.

The inspection process is indicated in figure 1 below, and follows Fagan/Gilb wrt. overall setup, duration etc.:

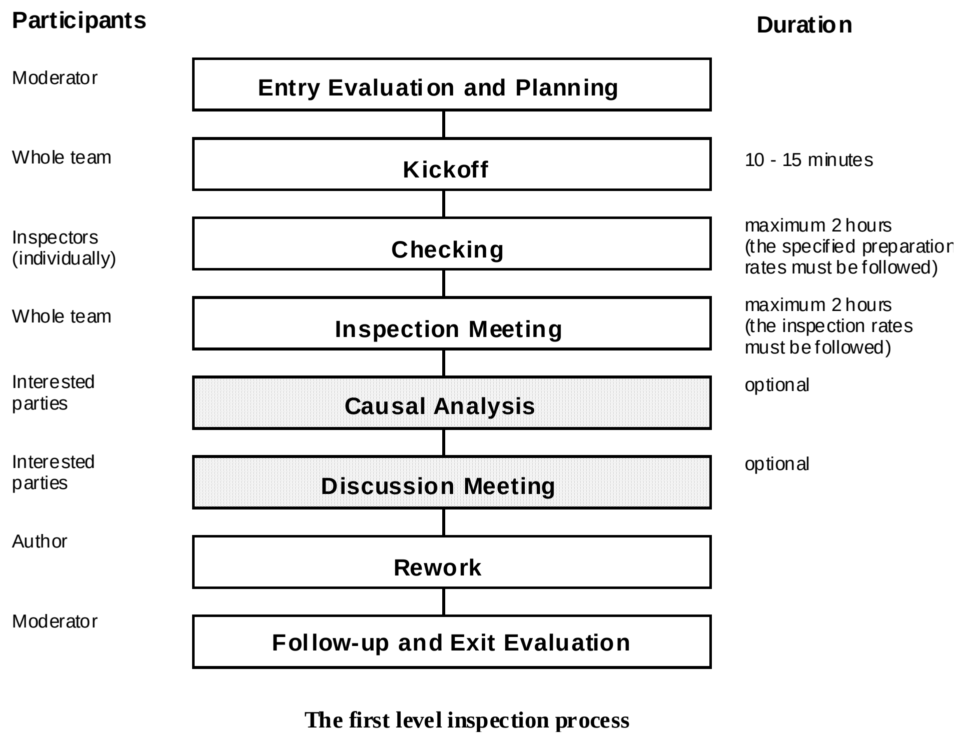

Participants
Duration
Moderator
Entry Evaluation and Planning
Whole team
Kickoff
10 - 15 minutes
Inspectors (individually)
Checking
maximum 2 hours (the specified preparation rates must be followed)
Whole team
Inspection Meeting
maximum 2 hours (the inspection rates must be followed)
Interested parties
Causal Analysis
optional
Interested parties
Discussion Meeting
optional
Author
Rework
Moderator
Follow-up and Exit Evaluation
The first level inspection process

Figure 1. Basic inspection process at Ericsson.

The different types of documents are presented in the table 1 below:

Table 1. Document types.

Document type	**Applicatin Information**
ADI	Adaptation Direction
AI	Application Information
BD	Block Description
BDFC	Block Description Flow Chart
COD	Command Description
FD	Function Description
FDFC	Function Description Flow Chart
FF	Function Framework
FS	Function Specification
FTI	Function Test Instruction
FTS	Function Test Specification
IP	Implementation Proposal
OPI	Operational Instruction
POD	Printout Description
PRI	Product Revision Information
SD	Signal Description
SPL	Source Parameter List
SPI	Source Program Information

4. Questions and hypotheses

4.1 Observation

O1: How cost-effective are inspections?

4.2 Questions

Q1: Are inspections performed at the recommended rates?

Q2: How cost-efficient are the inspection meetings?

Q3: Are the same kind of defects found in initial inspection preparations and following inspection meetings?

4.3 Hypotheses

The hypothesis is paired: one null hypothesis, $\mathbf{H}_0$, which is the one that will actually be tested and an alternative hypothesis, $\mathbf{H}_a$, which may be considered valid if the null hypothesis is rejected. For the statistical tests presented in this paper, a significance of level of 0.10 is assumed.

We will present two hypotheses. In each case, the null hypothesis will represents the positive conclusion, and the alternative hypothesis will conclude with the opposite. The three hypotheses are:

H1: Is there a significant, positive correlation between defects found during field use and document complexity?

H2: Is there a significant, positive correlation between defects found during inspection/test and module complexity?

Is there a significant correlation between defects rates across phases and deliveries for individual documents/modules? (i.e. try to track "defect-prone" modules).

5. Organization of the study

We have performed two studies where we have collected and analyzed historical data from software department at Ericsson in Oslo. Torbjørn Frotveit, our middleman at Ericsson, has all the time furmishing with the requested data.

This paper presents results from these two studies of inspection and testing:

- **Study 1:** This is the work done in a diploma thesis from 1997 [Marjara97]. Marjara investigated inspection and test data from Project A of 20,000 man-hours (14 man-years). Defect data in this work included inspection, desk check, function test, system test and partly field use.

- **Study 2:** This is the follow-up work done in the diploma thesis from 1998 [Skåtevik99]. This thesis has data from 6 different projects (Project A-F), including the project Marjara used in Study 1. It represent over 100.000 man-hours (70 man years). The test data in this work include only data from inspection and desk check, since later testings were done by other Ericsson divisions. However, it was possible to split desk check in code review and unit test, and data from these to activities are presented. Data from field use are not included, due to same reasons as for function- and system test.

Threats to internal validity:

We have used standard indicators on most properties (defect densities, inspection rates, effort consumption etc.), so all in all we are on agreed ground. However, wrt. Module complexity we are unsure, and further studies are needed. Whether the recorded defect data in the Ericsson database are trustworthy is hard to say. We certainly have discovered inconsistencies and missing data, but our confidence is pretty high.

Threats to external validity:

Since Ericsson has standard working processes worldwide, we can assume at least company-wide relevance. However, many of the findings are also in line with previous empirical studies, so we feel confident on general level.

6. The results and evalutation of these

This chapter presents the results from the two studies described in previous chapter, and tries to conclude the question and hypothesis stated in chapters 5.

Two definitions will be used throughout the chapter, effectiveness and efficiency:

Effectiveness: in finding defects regardless of cost.

Efficiency: cost-effective (as above per time-unit).

6.1 O1: How cost-effective are inspections?

In this section we shall describe, and compare the efficiency of inspections and testing at Ericsson in Oslo. Table 2 is taken from Study 1 and shows the effectiveness of inspections and testing, by comparing the number of defects found per hour.

Table 2. Total defects found, Study 1.

Activity	Defects [#]	[%]
Inspection preparation	928	61.8
Inspection meeting	29	1.9
Desk check (code review + unit test)	404	26.9
Function test	89	5.9
System test	17	1.1
Field use	35	2.3
Total	1502	100.0

Table 2 shows that inspections are the most effective verification activity, finding almost 64% of total defects found in the project. Second best is the desk check that finds almost 27%.

To analyze which of the verification activities that are most effective, the time spent on the different activities was gathered. Table 3 shows the time spent on the six verification activities.

Table 3. Cost of inspection and testing, defects found per hour, Study 1.

Activity	Defects [#]	Effort [h]	Time spent to find one defect [h:m]	Time spent on defect fixing [h]	Estimated saved time by early defect removal
Inspection preparation	928	786,8	00:51	311.2	8196.2
Inspection meeting	29	375,7	12:57		
Unit test	404	1257.0	03:07	-	-
Function test	89	7000.0	78:39	-	-
System test	17	-	-	-	-
Field use	35	-	-	-	-

When combining effort and number of defects, inspections proved to be the most cost-effective. Not surprisingly, function test is the most expensive activity. It should be noted that only human labor is included for desk check and function test. The costs of computer hours or special test tools are not included. Neither is the human effort in designing the test cases.

In Study 2 it was not possible to get defect data from function test, system test and field use. Instead the data made it possible to split up the desk check, which actually consist of code review and unit test. Table 4 shows the results.

Table 4. Total defects found, Study 2.

Activity	Defects [#]	[%]
Inspection preparation	4478	71.1
Inspection meeting	392	6.2
Desk check	832	13.2
Emulator test	598	9.5
Total	6300	100.0

Again, the data show that inspections are highly effective, contributing to 71.1% of all the defects found in the projects. Desk check is second best, finding almost 13% of the defects in the projects. Compared to Study 1, there is an improvement in the inspection meeting, whose effectiveness has increased from 2% to 6%.

Table 5 shows the effort of the different activities from Study 2.

Table 5. Cost of inspection and testing, defects found per hour, Study 2.

Activity	Defects [#]	Effort [h]	Time spent to find one defect [h:m]	Time spent on defect fixing [h]	Estimated saved time by early defect removal
Inspection preparation	4478	5563	01:15	11737	41467
Inspection meeting	392	3215	08:12		
Desk check	832	2440	02:56		-
Emulator test	598	4388	07:20		-

The inspection meeting itself has a much better cost efficiency in Study 2 (8h:12min per defect), compared to Study 1 (12h:57min per defect).

Although desk check on code seems to be the most effective method in Study 2, this method is not as general as inspections, which can be used on almost any phase/document of the process.

In Study 2, covering 100,000 man-hours, a total of 20515 hours were spent on inspections. It has been calculated that inspections did save 41467 hours, which would have been necessary to use on correcting defects otherwise found by testing. That is, saving of 41% of the total project effort.

Study 1 covered 20,000 man-hours where 1474 hours were spent on inspections. In this study it was calculated that they saved 8196 hours.

6.2 Q1: Are inspections performed at the recommended rates?

Here we want to see if the recommended inspection rates were applied in the inspections. The results are presented in table 6. Note that not all document types are included.

Table 6. Planned versus actual inspection-time consumption in Study 2.

Document information				Actual	Recommended time		Defects	
Document type	Number of documents	Total num. of pages	Average length of doc	Actual time	Planning constant	Recom. Time	Total number of defects	Defect density per page
ADI	1	7	7.00	36	20	20	12	1.71
AI	29	241	8.31	1019	72	2088	197	0.82
BD	41	1038	25.32	1438	40	1640	468	0.45
BDFC	54	3376	62.52	3531	104	5616	802	0.24
COD	4	31	7.75	105	14	56	38	1.23
FD	33	1149	34.82	2432	38	1254	784	0.68
FDFC	19	897	47.21	1230	26	494	338	0.38
FF	14	366	26.14	868	20	280	363	0.99
FS	14	244	17.43	950	24	336	205	0.84

FTI	2	605	302.50	216	14	28	22	0.04
FTS	2	154	77.00	840	14	28	44	0.29
IP	3	65	21.67	257	15	45	73	1.12
OPI	5	61	12.20	130	20	100	14	0.23
POD	4	23	5.75	116	20	80	29	1.26
PRI	57	582	10.21	1651	96	5472	399	0.69
SD	4	59	14.75	300	18	72	47	0.8
SPL	27	141	5.22	417	80	2160	69	0.49
Total	**313**	**9039**		**15536**		**19769**	**3904**	**0.43**

According to the recommended rates, the inspections are performed to fast (see table 6). 15536 hours are spent on inspections, whereas 19769 hours are recommended expenditure. The defect average per page is 0.43.

Study 1 also concluded with the same result, i.e. that inspections at Ericsson are performed to fast according to recommended inspection rates.

As reported in other literature, plots on preparation rate and defect detection rate (se figure 1) shows that the number of defects found per page decreases as the number of pages (document length) per hour increases. Inspection performed to fast will then results in decreased detection rate.

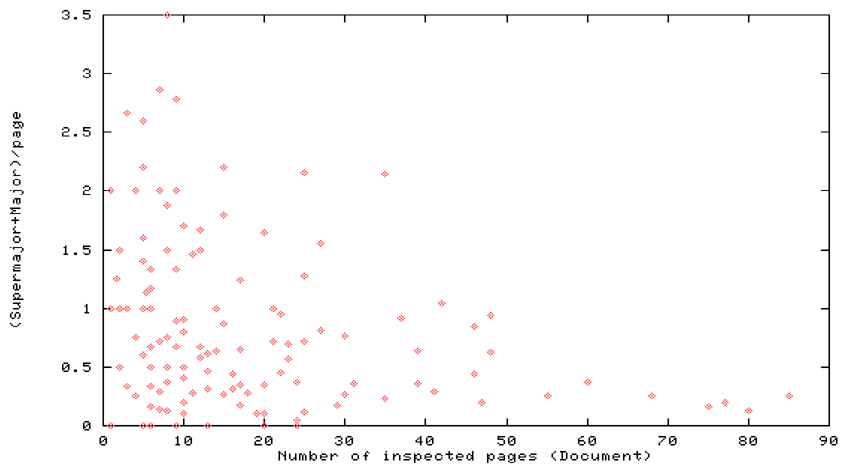


Figure 2. Number of pages inspected and defect detection rate, Study 1.

6.3 Q2: How cost-efficient are the inspection meetings?

Table 7 together with figure 3, shows the time consumption for each step of the inspections from Study 2. Effort before individual preparation and inspection meeting has here been proportionally distributed on these.

Table7. Time consumption for inspection, Study 2.

	Preparation	Inspection Meeting	Defect fixing	Sum
Hours	5563	3215	11737	20515
[%]	27.12 %	15.67 %	57.21 %	100,00%

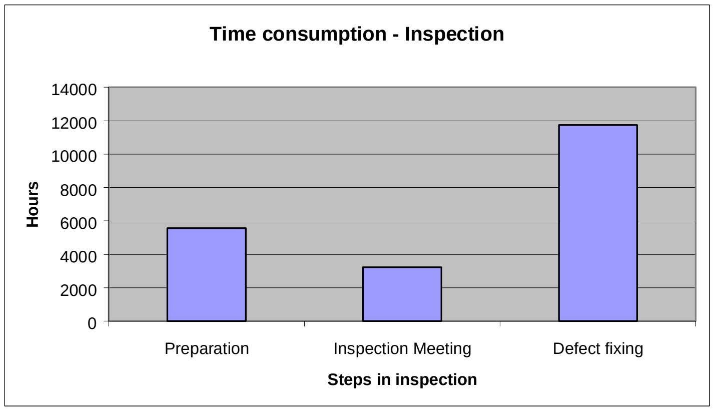


Figure 3. Time consumption for inspection, Study 2.

Table 8. Cost efficiency and defect classification from inspections, Study 2.

	Major		Super Major		Sum defects	Effort	Efficiency [defects/h]
	[#]	[%]	[#]	[%]	[#]	[h]	[defects/h]
Preparation	4356	97.2%	122	2.7%	4478	3415	1.31
In meeting	380	96.9%	12	3.1%	392	1956	0.20
In defect log	4736	97.2%	134	2.7%	4870	5371	0.91

Table 8 from Study 2, shows the number of defects recorded in preparations, in meetings, and the total. As mentioned, the defects are classified in two categories:

- **Major:** Defects that can have a major impact later, that might cause defects in the end products, and that will be expensive to clean up later.

- **Super Major:** Defects that have major impact on total cost of the project.

It turns out that 8% of defects found by inspections are found in meetings, with a cost-efficiency of 0.2 defects per hour. Compared to function test and system test, inspection meetings are indeed cost-effective in defect removal.

6.4 Q3: Are the same kind of defects found in initial inspection preparations and following inspection meetings?

We will also like to investigate what type of defects are found during preparations versus inspection meetings. Note: We do not have data on whether inspection meetings can refute defects reported from individual inspections (“false positive”), cf. [Votta93]. Our data only report new defects from inspection meetings (“true negative”). Table 8 from Study 2, shows that 2.7% of all defects from inspections are of type Super Major, while the rest are Major.

For preparation, the Super Major share is 2.7%. For meeting the share is 3.1%, i.e. only slightly higher. We therefore conclude that inspection meetings find the same “types” of defects as by individual preparation.

No such data were available in Study 1.

6.5 H1: Correlation between defects found during field use and document complexity

Intuitively, we would say that the faults in field use could be related to complexity of the module, and to the modification rate for the module. The modification rate indicates how much the module is changed from the base product, and the complexity is represented by the number of states in a module. For new modules the modification grade is zero. Correlation between modules and defect rates for each unit, (i.e., not the absolutely number of faults, but faults per volume-unit) has not yet been properly checked.

In Study 1, the regression equation can be written as:

$$N_{fu} = \alpha + \beta N_s + \lambda N_{mg}$$

where N_{fu} is number of faults in field use, N_s is number of states, N_{mg} is the modification grade, and α, β, and λ are coefficients. $\mathbf{H}_0$ can only be accepted if β and λ are significantly different from zero and the significance level for each of the coefficients is better than 0.10.

The following values was estimated:

$$N_{fu} = -1.73 + 0.084 * N_s + 0.097 * N_{mg}$$

Predictor	**Coefficient**	**StDev**	**t**	**P**
Constant	-1.732	1.067	-1.62	0.166
States	0.084	0.035	2.38	0.063
Modrate	0.097	0.034	2.89	0.034

The values for estimated coefficients are given above, along with their standard deviations, *t*-value for testing if the coefficient is 0, and the *p*-value for this test.

The analysis of variance is summarised below:

Source	**DF**	**SS**	**MS**	**F**	**P**
Regression	2	28.68	14.34	9.96	0.018
Error	5	7.20	1.44		
Total	7	35.88			

It should be noted that the coefficients are not significant, but that the states and modification rate are significant. The F-Fisher test is also significant, and therefore the hypothesis, $\mathbf{H}_0$ can be accepted based on the results from the regression analysis.

6.6 H2: Correlation between defects found during inspection/test and module complexity

The relevant data come from Study 2. Because just some of the modules are found over several lifecycles, only 12 modules out of 443 could be used for this analysis. 12 modules out of 443, shows that we should probably have checked out more thorougly relations between phases in same lifecycle, not just between different lifecycles.

Since data are collected for each document type, and each module in each phase consists of different number of document types, one document type is selected through all the phases. The document type selected is BDFC. Table 9 shows the results. Field marked with “-“ means that the data are missing, or no module exists. Because all the modules presented in this table only were included in project A through E, project F were excluded.

Table 9. Defect data for BDFC documents over different modules and projects, Study 2.

Module name	Project A			Project B			Project C			Project D			Project E		
	Def/page	Complexity	Defect found	Def/page	Complexity	Defect found	Def/page	Complexity	Defect found basic test	Def/page	Complexity	Defect found basic test	Def/page	Complexity	Defect found basic test
SUSAACA	0.04	72.0	25	0.28	80.5	3	-	-	-	-	-	-	-	-	-
SUSAACT	0.10	177.5	12	0.10	179.0	4	-	-	-	-	-	-	-	-	-
SUSCCTB	0.42	117.5	58	0.80	120.5	24	-	-	-	-	-	-	-	-	-
SUSCR	-	-	-	0.13	95.5	11	-	-	-	3.80	89.00	-	-	-	-
SUSCWC	0.29	-	23	0.10	-	-	-	-	-	-	-	-	-	-	-
SUSCWHF	-	-	11	0.50	-	-	-	-	-	-	-	-	-	-	-
SUSCWP	0.06	220.5	7	0.27	240.0	13	-	-	-	-	-	-	-	-	-
SUSSCR	0.08	244.5	22	-	295.5	34	-	-	-	-	-	-		-	-
SUSACF	0.14	47.0	32	0.37	62.5	28	-	-	-	-	-	-	0.24	66.0	-
SUSAP	0.26	67.0	42	-	-	10	-	-	-	-	-	-	0.04	78.0	-
SUSCCTA	0.34	269.5	118	-	297.5	132	1.00	299.5	3	-	-	-	-	-	-
SUSCS	0.06	257.0	14	0.90	267.5	34	0.18	254.5	21	-	-	-	-	-	-

Each project has data on defects per page found in inspections, the complexity of each module, and number of defects found in unit test for each block.

Hypothesis 2, uses the data presented above, and checks whether there exist a correlation between defects found during inspection/test and complexity for a module.

The regression equation used to state this hypothesis can be written as:

$Y = \alpha X + \beta$, where Y is defect density, X is the complexity and α, and β are constants.

H_0 can only be accepted if α and β are significantly different from zero and the significance level for each of the coefficients is better than 0.10.

The following values was estimated: Y = 0.1023*X + 13.595.

Table 10 shows the estimated values:

Table 10. Estimated values, Study 2

Predictor	*Estimate*	*Standard error*	*t*	*p*
β	13.595002	18.52051	0.73	0.4729
α	0.1022985	0.093689	1.09	0.2901

It indicates that the linear regression line must be rejected if a significance of level 0.10 is assumed, i.e. H_0 must therefore be rejected.

However, Ericsson reports that the best people often are allocated to develop difficult modules and more attention is generally devoted to complex software. This may explain why no significant correlation was found. Mores studies are anyhow needed here.

6.7 H3: Correlation between defects rates across phases and deliveries for individual documents/modules

This hypothesis, from Study 2, uses the same data as for hypothesis 2. To check for correlation between defect densities across phases and deliveries, we have analyzed the correlation between defect densities for modules over two projects. Because the lack of data in this analysis, only Project A and Project B where used (see table 9).

Table 11 shows the correlation results

Table 11. Correlation between defect density in Project A and B, Study 2.

Variable	**Defect density – Project A**	**Defect density - Project B**
Defect density – Project A	1.0000	0.4672
Defect density – Project B	0.4672	1.0000

With a correlation coefficient of 0.4672, we can not conclude that there exists a correlation between the two data set. We had only 6 modules with complete data for both projects for this test. The test should be done again, when a larger data set are available.

7. Conclusion

After analysis of the data, the following can be concluded for Ericsson, Oslo:

- Software inspections are indeed cost-effective:

  They find 70% of the recorded defects,

  cost 10% of the development time, and yield

  an estimated saving of 20%.

  I.e., finding and correcting defects before testing pays off, also here.
- 8% of the defects from inspections are found during the final meeting, 92% during the individual preparations. The same distribution of defects (Major, Super Major) are found in both cases. However, Gilb's insistency on finding many serious defects in the final inspection meeting is hardly true.
- The recommended inspection rates are not really followed: only 2/3 of the recommended time is being used.

- Individual inspections (preparations) and individual desk reviews are the most cost-effective techniques to detect defects, while system tests are the least effective.
- The final inspection meeting is not cost-effective, compared to individual inspections, in finding defects.
- The identified defects in a module do not depend on the module's complexity (number of states) or its modification rate, neither during inspections nor during testing.
- However, the defect density for one concrete system (Study 1) in field use correlated positively with its complexity and modification rate.
- We had insufficient data to clarify whether defect-prone modules from inspections continued to have higher defect densities over later test phases and over later deliveries.
- The collected, defect data has only been partly analyzed by Ericsson itself, so there is a huge potential for further analysis.
- The defect classification (Major and Super Major) is too coarse for causal analysis in order to reduce or prevent future defects, i.e. a process change, as recommended by Gilb. We also lack more precise data from Function test, System test and Field use.

It is somewhat unclear what these findings will mean for process improvement at Ericsson.

At least they show that their inspections are cost-effective, although they could be tuned wrt. recommended number of inspected pages per hour.

On the other hand, a more fine-grained data seem necessary for further analysis, e.g. for Root-Cause-Analysis (recommended by Gilb). Such defect classsification seems very cheap to implement at defect recording time, but is almost impossible to add later. However, Ericsson seems rather uninterested to pursue such changes, e.g. since "approval from headquarters" is necessary to modify the current inspection process.

Inspired by these findings, NTNU is anyhow interested to continue its cooperation with Ericsson on defect studies in the context of the SPIQ project. Their defect database seems underused, so these studies may encourage a more active utilization of collected data.

References

[Adams84]) Edward Adams:
"Optimizing Preventive Service of Software Products",
IBM Journal of Research and Development, (1):2--14, 1984.

[Basili96] Victor R. Basili, Scott Green, Oliver Laitenberger,
Filippo Lanubile, Forrest Shull, Sivert Sørumgård, and Marvin V. Zelkovitz:
"The Empirical Investigation of Perspective-Based Reading",
39 p., Empirical Software Engineering, 1996.

[Fagan76] Michael E. Fagan:
"Design and Code Inspection to Reduce Errors in Program Development",
IBM Systems J. Vol 15, No. 3, 1976.

[Fagan86] Michael E. Fagan:
"Advances in Software Inspections",
IEEE Trans. on Software Engineering, SE-12(7):744--751, July 1986.

[Gilb93] Tom Gilb and Dorothy Graham:
"Software Inspections",
Addison-Wesley, London, UK, 1993.

[Marjara97] Amarjit Singh Marjara:
"An Empirical Study of Inspection and Testing Data",
Technical report, NTNU, Trondheim, Norway, 22 Dec. 1997.

[Skåtevik99] Børge Skåtevik:
"An Empirical Study of Historical Inspection and Testing Data at Ericsson" (forthcoming),
Technical report, NTNU, Trondheim, Norway, 8 Feb. 1999.
ca. 150 p., EPOS TR 3xx (diploma thesis).

[Votta93] Lawrence G. Votta:
"Does Every Inspection Need a Meeting?"
In Proc. ACM SIGSOFT 93 Symposium on Foundation of Software
Engineering. ACM Press, December 1993.

A Process-Oriented Approach to Improving Software Product Quality

Richard E. Fairley
Professor and Director
Software Engineering Program
Oregon Graduate Institute
Beaverton, Oregon, USA

Abstract

Production of high quality software depends on early detection of defects and, better yet, prevention of defects. A process-oriented approach for improving defect detection and defect prevention by systematically collecting and analyzing defect data is described in this paper. The processes described here can be embedded in the various process models for software development; they are compatible with development models such as incremental, evolutionary, spiral, and Cleanroom. Because the processes described here are primarily those of software developers, they can be used in conjunction with other quality-enhancing processes such as quality assurance and independent verification and validation.

1. Introduction

The processes used to accomplish the work activities of software engineering are important factors in software productivity, project predictability, and the quality of the work products produced by software engineers. Quality factors for software vary widely depending on the application domain and the needs of users. Safety is the most important quality factor in software-intensive systems that involve risk to human life; for example, the on-board software of the NASA Space Shuttle, the control system of a nuclear reactor, or a medical system. Security is the most important quality attribute in financial transaction systems, while reliability and availability may be most important for telecommunication systems. Ease of learning and ease of use are important quality attributes for systems involving human-computer interaction; however ease of learning and ease of use will be regarded differently by people having different backgrounds and skill levels.

Lack of a necessary or desired attribute in a software system or software product is caused by defects (or faults) created in the work products generated during initial development or subsequent modification of the system. Defects result when something is left out, when something is done wrong, or when something unnecessary is added to software. A product failure results when a defect is encountered during operation of a system. Different types of defects result in different types of failures. A system crash is the result of one or more defects in the system, and "hard to use," as judged by the target population of a human-computer system, is also the result of defects in the system. Defects are thus those attributes of a system that cause departures from specified or desired behavior.

Software defects are created when humans make mistakes of omission (leave something out) and mistakes of commission (doing something wrong or something extra). In software engineering, human mistakes are the result of faulty communication and coordination processes, lack of sufficient time to do the job correctly, lack of adequate skills and tools, poorly designed products that are difficult to modify, and human fallibility. Some software defects are inevitable because humans are not infallible, our skills and tools are not perfect, our systems (especially legacy systems) are often overly complex, our schedules are often unrealistic, and our processes of communication and coordination are not perfect.

Production of high quality software depends on early detection of defects and, better yet, prevention of defects. A method for improving defect detection and defect prevention by systematically collecting and analyzing defect data is described in this paper. The processes described here can be embedded in the various process models for software development; they are compatible with development models such as incremental, evolutionary, spiral, and Cleanroom [1]. Because the processes described here are primarily those of software developers, they can be used in conjunction with other quality-enhancing processes such as quality assurance and independent verification and validation.

Section 2 of this paper describes the model. Techniques for defect analysis are presented in Section 3, considerations for defect prevention are presented in Section 4, and Section 5 presents issues related to measuring the cost of software quality. Process enactment is discussed in Section 6 and the summary and conclusions are in Section 7.

2. A Process for Recording Defects and Tracking Rework

A model for recording software defects and tracking rework effort is presented in Figure 1. The term "rework" means effort spent fixing defects (correcting mistakes). As illustrated in Figure 1, a work product is the private property of the author(s) until it passes some pre-determined acceptance criteria; it then becomes public property of the work group. Mistakes discovered and corrected in a private work product by the developer are not counted as defects. Defects are counted and rework is tracked when mistakes are found in work products during application of the acceptance criteria or after they pass their acceptance criteria and become public work products.

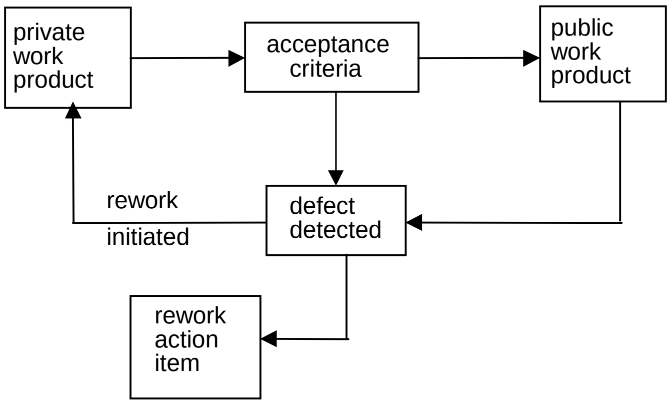


Figure 1. A Defect and Rework Tracking Model

A "work product" is any artifact generated or modified by one or more members of a project team that becomes public property of the project. To become a public work product, an artifact must be evaluated and accepted by two or more team members. Examples of work products include (entire or partial) operational concept documents, requirements specifications, project plans, design documents, interface specifications, traceability matrices, test plans, test procedures, code modules, configuration management reports, quality assurance audits, and so forth. Examples of acceptance criteria for various types of work products are provided in the Appendix to this paper. A universal acceptance criterion applied to all work products is that the intended receivers of a work product must agree that the work product is an acceptable input to their work processes or that the work product is acceptable for delivery to customers and users.

All work products have the potential to contain defects and are thus candidates for acceptance and control as illustrated in Figure 1. A complete artifact may be generated in several steps or stages, so that the work products cited in Figure 1 may be small units of output generated and evaluated on a frequent (perhaps weekly) basis. A work product, large or small, is not accepted until the associated quality criteria are satisfied. Upon acceptance, the electronic file containing the work product is placed under version control and cannot be subsequently changed without agreement to do so by the affected parties. Work products controlled in this manner are said to be under baseline control.

A version control system that includes mechanisms and procedures for check-in, check-out, and controlled updating of work products is an important element of this model. Once checked in, a work product is checked out in response to a change in the requirements or to fix a defect (i.e., to correct a mistake). Satisfying the quality criteria for a work product does not guarantee that the work product is defect-free but that the quality is thought to be acceptable, as determined by the criteria applied. If, at some later time a defect is found in an accepted work product, an action item is generated and rework is accomplished to correct the defect. When the defect is corrected, the work product is checked in to the version control system, a new version of the work product is generated, and all affected parties are notified of the change. This process is illustrated in Figure 2.

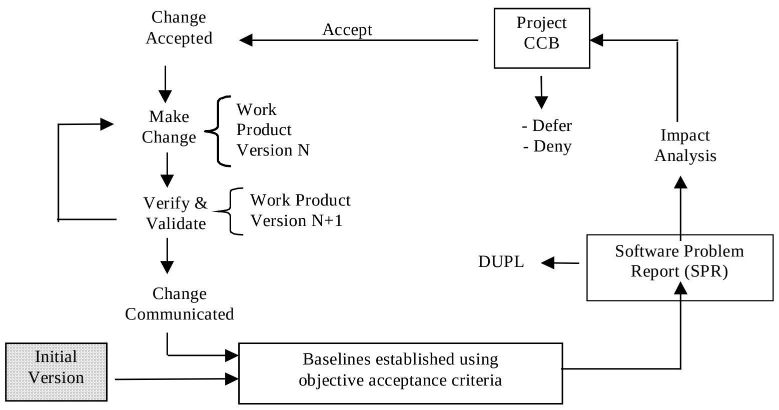


Figure 2. The Change Control Process

The format of a rework action item is presented in Table 1. Rework action items are entered into a tracking system, progress is reviewed periodically (perhaps weekly), and each action item is tracked to closure. An action item is closed when the corrected work product passes its acceptance criteria and a new version of the work product is checked into the version control system. The problem resolution system is thus a closed-loop, as required by ISO/IEC Standard 12207 [2]. A status report of rework in progress, as illustrated in Table 2, is a convenient way to periodically report the status of on-going action items; each rework action item that is in progress has a corresponding entry in the status report.

Table 1. Format of a Rework Action Item

1. Action Item Number and Name
2. Defect Category
3. Actions to be taken: planned & actual
4. Responsible party: planned & actual
5. Resources applied: planned & actual
6. Milestones and dates: planned & actual
7. Completion criteria: planned & actual
8. Completion date: planned & actual

Table 2. Rework-in-Progress Report

Date: 1 August 199x

Item Number	Rework Issue	Scheduled Completion	Estimated Completion
3.1.2	Correct the interface between the VS and EXEC modules	12 Aug	12 Aug
4.2.3	Synchronize the IN2 interrupt and BUFFER routine	5 Aug	10 Aug
2.2.5	Correct the SIZE routine in the WBS data abstraction	7 Aug	4 Aug
5.1.6	Fix the CURSOR routine exit sequence	3 Aug	5 Aug
3.1.5	Change the navigation sequence of user screens	7 Jul	TBD

3. Analysis of Defect Data

Patterns of defect creation and detection and the rework effort required to fix defects can be captured using the process illustrated in Figure 1 and Table 1. Defect data is compiled from completed rework action item reports (Table 1) and presented using the formats of Tables 3 and 4. In Table 3, for example, the percent of requirements defects detected during requirements work activities can be calculated as: (RDr * 100) / RDt; the percent of design defects that escape design work activities is: [1– (DDd / DDt)] * 100]; and the percent of total defects detected by users is: (OSt * 100) / TOTAL.

Table 3. Defects by Category

Defect Type: \ Work Activity:	Rqmts	Design	Code	Test	Ops	Totals
Rqmts	RdR	RdD	RdC	RdT	RdO	ΣRd
Design		DdD	DdC	DdT	DdO	ΣDd
Code			CdC	CdT	CdO	ΣCd
Test				TdT	TdO	ΣTd
Totals	ΣdRa	ΣdDa	ΣdCaA	ΣdTa	ΣdOa	TOTAL

Legend:

XdY:	X is type of defect: R = Requirements; D = Design; C = Code; T = Test d is defect Y is type of work activity in which defect is detected; Y is one of R = Requirements; D = Design; C = Coding; T = Testing; O = Operation
ΣXd:	total defects of type X; X is one of R, D, C, or T
ΣdYa:	total defects of all types found during work activities of type Y

Tables in the form of Table 4 are used to report the amount of rework effort required to fix defects of various types at various stages of work. The percentage distributions of values in Tables 3 and 4 may be different, which would indicate that some types of defects require more rework effort than other types of defects.

The information contained in Tables 3 and 4 can provide guidance for defect containment and defect prevention activities. As is well-known, it becomes more expensive to fix a defect the longer the time between defect creation and defect detection [4]. For example, early detection and correction of requirements defects and design defects is desirable because of the "amplification effect" whereby two requirements defects might result in five design defects and twenty code defects if not detected at the requirements stage. From the pattern of rework effort in Table 4, it might be observed that the most effective place to focus defect containment activities is in the design activity. Defect containment activities might include better documentation and more extensive inspection of interfaces, prototyping of algorithms, more attention to traceability, and so forth. Information concerning a corrected defect should be entered in Tables 3 and 4 before a rework action item is closed.

Table 4. Rework by Category

Defect Type: \ Work Activity:	Rqmts	Design	Code	Test	Ops	Totals
Rqmts	RrR	RrD	RrC	RrT	RrO	ΣRr
Design		DrD	DrC	DrT	DrO	ΣDr
Code			CrC	CrT	CrO	ΣCr
Test				TrT	TrO	ΣTr
Totals	ΣrRa	ΣrDa	ΣrCa	ΣrTa	ΣrOa	TOTAL

Legend:

XrY:	X is type of rework: R = Requirements; D = Design; C = Code; T = Test r is rework Y is type of work activity in which rework occurs; Y is one of R = Requirements; D = Design; C = Coding; T = Testing; O = Operation
ΣXr:	total rework of type X; X is one of R, D, C, or T
ΣrYa:	total rework of all types during work activities of type Y

4. Defect Prevention

It is better to prevent defects than to detect and correct them, for two reasons: 1) to reduce rework, thereby increasing productivity and 2) to deliver higher quality products to customers and users. In many organizations, the level of rework exceeds 25% (and in some cases 50%) of all software engineering work activities [3], [4]. In a well-known case study, reported in [3], Raytheon Corporation reduced rework from 41% in 1988 to 11% in 1992. An investment of $1.8M (U.S.) resulted in a savings of $15.8M (U.S.) during the four year period (and the savings continue into the future). Better defect detection and prevention results in higher quality products delivered to users because defect containment prior to product release is never 100% effective. If we create 20 defects per thousand lines of code and our defect containment effectiveness is 90%, we will release 2 defects per thousand lines of code to the users. If we create 2 defects per thousand lines of code and our defect containment effectiveness is 99%, we will release 0.02 defects per thousand lines of code. The former example is typical of many organizations; the latter is typical of safety-critical systems such as the Space Shuttle on-board software [5].

Each rework report, as illustrated in Table 1, categorizes the type of defect corrected. For example, a requirements defect might be categorized as being incomplete, incorrect, inconsistent, or ambiguous; while a code defect might be categorized as an interface, logic, computation, data definition, or data usage defect. Periodic review of rework action-items is accomplished in causal analysis meetings [6]. Causal analysis will often reveal common, underlying reasons for similar defects. If, for example, a large number of interface defects are found in the design documentation, steps must be taken to improve the quality of the interface specifications. Corrective actions to be taken might include changing the notation used to document interfaces, improving the development infrastructure (hardware platform, software environment, physical facilities), acquiring new software tools, modifying development methods and procedures, and/or training of personnel.

5. Measuring the Cost of Quality

In many organizations, the level of rework associated with defect detection and correction is a major source of inefficiency (non-productive use of time and effort) and ineffectiveness (failure to achieve the desired result). In these organizations, improvement in early detection and prevention of defects more than pays for the investment required - as in the Raytheon case - by reducing rework and thus improving the efficiency and effectiveness of the software development process. In a 1995 report, an SEI study of process improvement efforts among 13 organizations reported an average return on investment of 5 to 1, with a range between 5 to 1 and 10 to 1 [7]. These efforts have clearly returned more than invested to improve quality and productivity.

On the other hand, safety critical systems such as the NASA Space Shuttle's on-board software, have quality requirements so stringent that defect detection and prevention activities, albeit very efficient, are so extensive that they increase the time, effort, and cost of software development and modification. There is thus the extreme of insufficient quality, which increases the time, effort, and cost for rework and the extreme of extraordinary quality, which increases the time, effort, and cost required to achieve the necessary level of quality.

This "bathtub" relationship is illustrated in Figure 3.

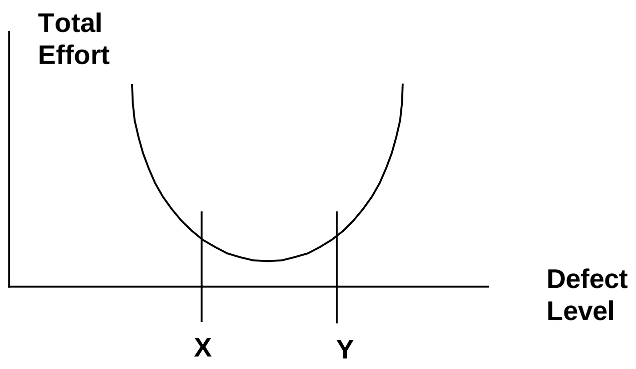


Figure 3. Defect Level and Total Effort

As portrayed in Figure 3, delivered defect densities greater than "Y" provide opportunities for cost savings by investing in quality improvement (rework reduction). Achieving defect densities less than "X" requires increased effort to achieve the desired level of quality. It is apparent that every software development organization that experiences defect levels greater than "Y" should invest in improvements to their defect detection and defect prevention processes. Experience shows that these activities will result in cost savings, schedule reduction, and quality improvement (see [3]-[7]). Each development organization and user community must decide whether defect levels less than "X" justify the increased cost of quality associated with those defect levels.

In many organizations, we have observed values of "X" and "Y" in the range of X = 0.1 and Y = 1 delivered defects per thousand lines of code, where delivered defect density is measured as defects per thousand lines of code reported by users during the first year of using the software. The data needed to determine appropriate values of X and Y for a software development organization can be obtained using the processes described in this paper.

6. Process Enactment

Enactment of the quality improvement process described here requires that several roles be played in a software organization. These roles include those to determine acceptance criteria for various types of work products; those to generate work products; those to accomplish rework to fix work products; those to apply the acceptance criteria; those to enact version control; those to collect, validate, record, and analyze defect levels and rework activities; those to recommend process improvements; and those to implement process improvements.

Depending on the size and complexity of software activities within an organizational unit, one person may play several roles, either concurrently or serially, or one role may require several people for enactment. Titles assigned to quality-process roles may include work product generator, work product acceptor, version controller, work product reworker, data collector/validator/recorder, rework analyzer, and process improver. Tasks that must be accomplished to enact quality processes include development of acceptance criteria for the various types of work products, and development of acceptance mechanisms (such as peer reviews, analysis, testing, and demonstration of work products); in addition, version control mechanisms and procedures must be developed if they do not exist.

A typical organizational structure for software projects that adopt this model is illustrated in Figure 4. As illustrated there, large projects are organized around small teams. Each small team has a team leader and 3 to 5 team members. Requirements for component features, interfaces, performance, memory usage, and so forth are allocated to each small team. The product structure is thus embedded in the structure of the project team that develops the product [8]. Each work product developed by each small team is subject to pre-determined acceptance criteria (see the Appendix). The team leader of each small team is responsible for ensuring that all work products generated by the team satisfy the appropriate acceptance criteria. Keeping teams small makes it possible for the team leader to serve as the first agent of quality control of work products. The software architect for the project works with the team leaders to ensure that decisions affecting multiple components and the interfaces among components preserve the quality criteria for the software product.

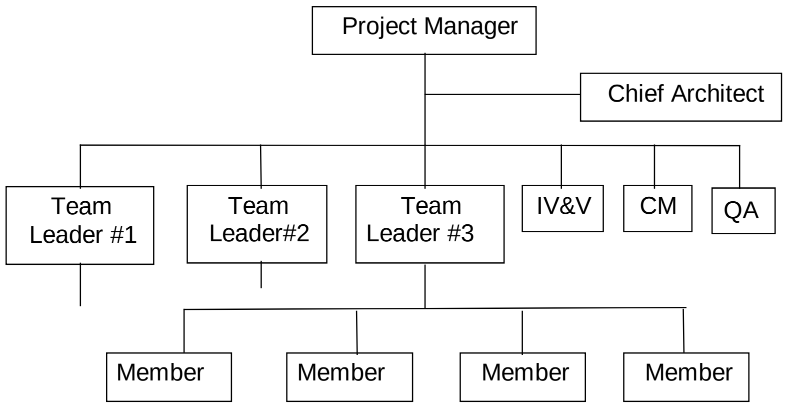


Figure 4. Structure of a Software Project

7. Summary and Conclusions

Software defects are the result of mistakes made by individuals and teams of individuals engaged in intellectual work activities that require close coordination. People make mistakes because of failures in communication and coordination; because of insufficient time to accomplish the work; because of inadequate methods, tools, training, and experience; because of difficult-to-modify legacy systems, and because people are human and subject to human fallibility. The purpose of process improvement is to reduce the chance that people will make mistakes, thereby improving software quality, reducing software cost, and delivering software in less time within more predictable schedules and budgets.

We cannot expect to produce defect-free software, but we can reduce the opportunities to make mistakes by improving our work processes. This involves learning from our mistakes, improving communication and coordination, and applying appropriate methods and tools in a work environment where work-related stress is reduced, thus reducing mistakes caused by human fallibility.

Rework (non-productive work) must be expended to correct detected defects. Rework can be reduced by early detection of defects and by revising the work processes that result in defect creation. This paper has presented a systematic process to reduce rework in several ways:

1. Predetermined acceptance criteria are applied to each work product. Work products are generated in small increments, as illustrated in Figure 1, thus controlling the number of defects that escape initial development or modification of a work product.
2. Accepted work products are placed under version control, thus protecting them from uncontrolled change and providing a defect tracking and notification-of-change mechanism.
3. A rework action item is opened when a defect is detected in an accepted work product, thus providing a mechanism to track rework effort.
4. Rework action items are tracked to closure, thus ensuring that rework is completed in a closed-loop problem resolution system.
5. A corrected work product is not accepted until it satisfies the associated acceptance criteria and a rework action-item report is completed, thus ensuring that the defect has been fixed and that sufficient information has been recorded to allow causal analysis of defects.
6. When a corrected work product is accepted, a new version number is generated and all affected parties are notified of the change, thus ensuring sufficient communication among team members and coordination of related work activities.
7. Trends in defect creation, detection, and correction are identified using the information in Tables 1-4, thus providing indicators of candidate areas for process improvement.
8. Causal analysis meetings are held periodically (perhaps monthly) to determine the underlying causes of similar defects and to recommend process improvement initiatives that should be undertaken.
9. Process improvement initiatives are planned, monitored, and tracked to closure as process improvement action items.
10. The quality measurement system is used to measure and evaluate the results of process improvement initiatives, thus providing a closed loop quality improvement system.

Quality improvement is a never-ending task. There will always be more efficient ways and more effective ways to accomplish the complex tasks of developing and modifying software artifacts - the intellectual work products generated by teams of individuals engaged in work activities that must be closely coordinated. The processes described in this paper provide some mechanisms that can be used to improve software product quality.

References

[1] S.L. Pfleeger, *Software Engineering Theory and Practice*, Prentice Hall, 1998.

[2] ISO/IEC12207-1-1994, *Software lifecycle processes.*

[3] R. Dion, "Process improvement and the corporate balance sheet," *IEEE Software*, vol. 10, no. 4, July, 1993.

[4] B.W. Boehm and C. Papaccio, "Understanding and controlling software costs," *IEEE Trans. On Software Engineering*, vol. 14, no. 10, Oct. 1988.

[5] C. Billings et al, "Journey to a mature software process," *IBM Systems Journal*, vol. 33, no. 1, 1994.

[6] R. Mays et al, "Experiences with defect prevention," *IBM Systems Journal*, vol. 29, 1990.

[7] J. Herbsleb et al, *Benefits of CMM-Based Software Process Improvement: Initial Results*, SEI-CMU-94-TR-13, Software Engineering Institute, Pittsburgh, USA

[8] M.E. Conway, "How Do Committees Invent?," *Datamation*, April, 1968.

Appendix

SOME ACCEPTANCE CRITERIA FOR SOFTWARE WORK PRODUCTS

• Requirements
- traced to user needs/scenarios/use cases
- inspected by peers
- validation tests / test scenarios generated
- signed-off by customer

• Design Documentation
- traced to requirements
- inspected by peers
- interfaces verified
- exception handling adequate
- sizing and timing analyzed
- associated integration tests generated
- signed-off by Chief Architect

• Source Code Modules
- traced to requirements and design
- inspected by peers
- unit tested
- signed-off by team leader
- interfaces inspected
- traced to executed integration tests
- performance verified
- signed-off by Configuration Manager

• Integrated Software Subsystem
- interfaces inspected
- traced to executed integration tests
- performance verified
- signed-off by Configuration Manager

• Integrated System (hardware/software)
- traced to executed system tests
- hardware interfaces exercised
- performance verified
- exception handling verified
- sign-off by system engineering or independent testing

• Documentation
- inspected by peers
- validated by independent parties
- signed-off by independent evaluation group

(Documentation includes: end-user documentation,
principles of operation, as-built design specifications,
maintenance guides)

Quality first

Measuring a safety-critical embedded software development process

E. Kesseler
National Aerospace Laboratory NLR
kesseler@nlr.nl

Amsterdam, Netherlands

Abstract

Software which is embedded in aircraft to which people entrust their lifes becomes safety-critical and consequently must be of the highest quality. Failures of such software must be virtually non-existent. Due to the high costs of aircraft, hypothetical software failures would also incur major financial losses. To guarantee that the safety of aircraft is an uncompromisable requirement, an independent government agency certifies aircraft as fit-for-use.

The experience with a software development process model accommodating both safety-critical requirements as well as commercial requirements is described. The findings are based on process and product metrics. The first two versions of the software product have successfully passed the certification and are currently being flown in numerous aircraft. In fact the software product is so successful that it will be adapted to other aircraft models.

Measuring the requirements evolution contributed to the change from a waterfall based software development process to a spiral software development process. Design stability is refelected in the module evolution but needs to be complemented by other information. Requirements evolution and its implementation status combined with design stability help in the trade-off between additional deliveries, their functions and their release dates.

1. Introduction

Software which is embedded in aircraft to which people entrust their lifes becomes safety-critical and consequently must be of the highest standards. Failures of such software must be so rare as virtually non-existing during the life time of all aircraft concerned. Due to the high costs of aircraft, hypothetical software failures would also incur major financial losses, a further drive to require the highest quality. It is clear that in aircraft safety is an uncompromisable requirement.

To guarantee the safety of aircraft, an independent government agency certifies aircraft as fit-for-use. Only after this certification the aircraft may be used commercially. To guarantee a world-wide equal level of safety, software airworthiness requirements are stated in one document, [DO-178B]. This document contains information for both the certification authorities and the developers.

A software development process based on the waterfall model is a well-proven way to produce safety-critical software. [DEKK, KESS] provides an example where an ESA-PSS05 compliant process is used. For complex technical systems like aircraft, the commercially determined time-to-market mandates co-development of the various subsystems. Co-development will inevitably result in requirements evolution. Even more so if complicated Human Machine Interfaces are involved. The waterfall model is not intended to cope with such requirements evolution.

The experience with a DO-178B compliant software development process which accommodates a commercial time-to-market is described. The findings are based on process and product metrics. The first two product versions have successfully passed the certification and are currently being flown in numerous aircraft. In fact the software product is so successful that it will be adapted to other aircraft models.

The sequel starts with a short description of the application. Subsequently some information about the air transport safety requirements is provided, together with its influence on the software development process to be applied. The experience gained during the production of the embedded application is described, supported by metrics. The findings are summarised in the conclusions.

2. Application description

To fly aircraft under all (adverse) conditions, pilots must fully rely on the data presented to them, and on the correct and timely forwarding of their commands to the relevant aircraft subsystems. The embedded avionics application discussed combines, controls, processes and forwards the data between the subsystems and the flight deck. The flight deck may contain conventional mechanical displays or a modern Electronic Flight Instrument System (EFIS) or even a mix of these. The application generates all information for the flight deck as well as processes all pilot inputs. This renders the application vulnerable to changes in the aircraft's Human Machine Interfaces.

The embedded application is designed to operate in both Visual Meteorological Conditions (VMC) and Instrument Meteorological Conditions (IMC). In the former conditions, the pilot can obtain part of the necessary flight information from visual cues from outside the cockpit. These conditions limit the aircraft operations to good weather operations. The latter conditions allow all-weather operations of the aircraft. Under these conditions the displays of the flight deck are needed by the pilot to fly. This renders the correct functioning of the displays safety-critical. A number of equipment items needs to be duplicated to achieve the required low failure probability.

During normal operations the embedded application processes about 100 different flight parameters, originating from 10 different sensors, some of which are duplicated. Two processors are used in each of the duplicated hardware units. The delay times within the entire embedded application should be guaranteed to be less then 30 milliseconds with a cycle time of 25 milliseconds for the main processor. During the operational life of the embedded application many extensions are expected, so 50% spare processor time shall be allowed for. The I/O processor has a cycle time of 360 microseconds.

The influence of safety on the embedded application's functions will be illustrated for data input. Depending on the criticality of the flight parameter, the software validates it in up to four complementary ways:

- coherency test: a check on correct length and parity of the data;
- reception test: a check on the timely arrival of the data;
- sensor discrepancy test: a comparison between the two data values produced by the two independent redundant sensors; and
- module discrepancy test: a comparison between the two parameter values produced by the same sensor; one value directly read by the system from the sensor, and one obtained from the redundant system via a cross-talk bus.

[Kess, Slui] contains more information on the application.

3. Air transport safety requirements

3.1 Applicable software safety document

For safety-critical software in airborne equipment [DO-178B] has been developed. This document provides guidance for both the software developers and the certification authorities. In civil aviation an independent governmental institution, the certification authority, performs the ultimate system acceptance by certifying the entire aircraft. Only then the constituent software is airworthy and ready for commercial use. [DO-178B] provides a world-wide "level playing field" for the competing industries as well as a world-wide protection of the air traveller, which are important due to the international character of the industry. In NLR's case the certification authority concerned delegated some of its technical activities to a specialised company.

Certifying the entire aircraft implies that when an aircraft operator wants an aircraft with substantial modifications, the aircraft including its embedded software has to be re-certified. Substantial modifications are, for example, modifications which can not be accommodated by changing the certified configuration files.

[DO-178B] was the first widely used document to address safety- critical software. Based on amongst others the experience gained with this document, currently other more general purpose standards are available, like [ISO/DIS 15026] and [IEC 61508]. [SAE ARP 4754] addresses the certification considerations for highly-integrated or complex aircraft systems. [SAE ARP 4754] is complementary to [DO-178B] and applicable hardware specific standards.

3.2 Safety classification

Based on the impact of the system (i.e. aircraft) failure the software failure can contribute to, the software is classified into 5 levels. The failure probability in flight hours (i.e. actual operating hours) according to the Federal Aviation Requirements /Joint Aviation Requirements [FAR/JAR-25] has been added. [FAR/JAR-25] uses the general principle of an inverse relationship between the probability of a failure condition and the degree of hazard to the aircraft or its occupants. As [DO-178B] considers qualitative demonstration of software compliance to such high reliability to be beyond the current software technology, the [FAR/JAR-25] numbers are provided for information only.

Level A: Catastrophic failure

Failure conditions which would prevent continued safe flight and landing.

[FAR/JAR-25] extremely improbable, catastrophic failure < $1x10^{-9}$

These failure conditions are so unlikely that they are not anticipated to occur during the entire life of all aircraft of one type.

Level B: Hazardous/Severe-Major failure

Failure conditions which would reduce the capability of the aircraft or the ability of the crew to cope with adverse operating conditions to the extent that there would be:

- a large reduction in safety margins or functional capabilities;
- physical distress or higher workload such that the flight crew could not be relied on to perform their tasks accurately or completely;

- adverse effect on occupants including serious or potentially fatal injuries to a small number of those occupants.
  [FAR/JAR-25] extremely remote, $1x10^{-9}$ < hazardous failure < $1x10^{-7}$

Level C: Major failure

Failure conditions which would reduce the capability of the aircraft or the ability of the crew to cope with adverse operating conditions to the extent that there would be, for example,

- a significant reduction in safety margins or functional capabilities;
- a significant increase in crew workload or in conditions impairing crew efficiency, or
- discomfort to occupants, possibly including injuries.
  [FAR/JAR-25] remote, $1x10^{-7}$ < major failure < $1x10^{-5}$

Level D: Minor failure

Failure conditions which would not significantly reduce aircraft safety and which would involve crew actions that are well within their capabilities. Minor failure conditions may include for example,

- a slight reduction in safety margins or functional capabilities;
- a slight increase in crew workload, such as, routine flight plan changes or some inconvenience to occupants.
  [FAR/JAR-25] probable, minor failure > $1x10^{-5}$

Level E: No Effect

Failure conditions which do not affect the operational capability of the aircraft or increase crew workload.

3.3 Software life cycle

[DO-178B] deliberately refrains from making statements about appropriate software life cycle models. The life cycle is described rather abstract as a number of processes that are categorised as follows:

- software planning process which entails the production of the following documents:
  - plan for software aspects of certification. The main purpose of this document is to define the compliance of the chosen software development process to [DO-178B] for the certification authorities. This document contains many references to the project documentation generated as part of the applied life cycle model;
  - software development plan, which defines the chosen software life cycle and the software development environment, including all tools used;
  - software verification plan, which defines the means by which the verification objectives will be met;
  - software configuration management plan;
  - software quality assurance plan.
- software development processes consisting of :
  - software requirement process;
  - software design process;
  - software coding process;
  - integration process.
- integral processes which are divided into :

- software verification process;
- software configuration management process;
- software quality assurance process;
- certification liaison process.

The integral processes are a result of the criticality of the software. Consequently the integral processes are performed concurrently with the software development processes throughout the entire software life cycle.

3.4 Verification

In order to provide the developer with maximum flexibility, [DO-178B] allows the developer to choose the software life cycle model. [DO-178B] enforces traceability to its general requirements by verifying that the life cycle process provides all data it requires. Each constituent software development process has to be traceable, verifiable and consistent. Transition criteria need to be defined by the developer to determine whether the next software development process may be started. In case of iterative processes, like in the spiral model, attention needs to be paid to the verification of process inputs which become available after the subsequent process is started.

Verification is defined in [DO-178B] as "the evaluation of the results of a process to ensure correctness and consistency with respect to the inputs and standards to that process". Verification can be accomplished by review, analysis, test or any combination of these three activities. Review provides a qualitative assessment of correctness.

Analysis is a detailed examination of a software component. It is a repeatable process that can be supported by tools. Every tool needs to be verified against the Tool Operational Requirements, the contents of which is prescribed in [DO-178B]. For software tools the same documentation and configuration control procedures apply as for the airborne software. Every software tool needs approval of the certification authority.

Testing is "the process of exercising a system or system components to verify that it satisfies specified requirements and to detect errors". By definition the actual testing of deliverable software forms only part of the verification of the coding and integration processes. For software classified at [DO-178B] level A, a mandatory 100% code coverage applies. This code coverage consists of :

- statement coverage (every statement executed, called statement testing in [BS7925-2]);

- decision coverage (every decision executed for pass and fail, called branch/decision testing in [BS7925-2]), and

- the modified condition/ decision coverage (mc/dc, same name in [BS7925-2]). Mc/dc requires that for every condition in a decision, its effect on the outcome of the decision is demonstrated.

Code coverage will be shown at module level testing.

4. Software development process

The definition of the software development process has been guided by previous experience with safety-critical software for spacecraft. More information on the spacecraft application is provided in [Dekk, Kess].

The project team was set up consisting of 2 separate groups, a development group and a verification group. The verification group was headed by a team member with sufficient authority to report, at his own discretion, to the company management outside the project hierarchy, in compliance with [DO-178B]. Furthermore the quality assurance manager was independent from both teams and not allowed to produce deliverable code or tests. The quality assurance manager needed his technical background in order to judge technical choices made.

The embedded application project started using :

- the DOD-STD-2167A life cycle model [DOD], which is based on the waterfall model ;

- customer supplied requirement specifications in plain English ;
- formal reviews after each life cycle phase;
- software analysis using Structured Analysis with Hatley and Pirbhai Real Time extensions (SA/RT) [Hatl, Pirb] supported by a Computer Aided Software Engineering (CASE) tool;
- software design using Yourdon Structured Design (SD) supported by the same CASE tool;
- the customer prescribed C language;
- NLR's proprietary C coding standard, with project specific enhancements and enforced by a static analysis tool;
- execution of module tests and integration tests on the target system;
- an automated test tool to aid the construction and cost effective repetition of the functional tests and code coverage tests;
- a proprietary configuration management tool;
- module and integration testing on the target with a simulated environment;
- integration with the aircraft avionics suite after integration of the embedded application.

5. Experience

5.1 DO-178B experience

Modern aircraft contain huge amounts of software, supplied by numerous independent suppliers world-wide. Even a single aircraft contains software of many different suppliers. According to the US National Transport Safety Board (NTSB), [DO-178B] works well as up to now no catastrophic failure (i.e. fatalities or hull losses) can be directly attributed to a software failure [IEEE]. An independent software engineering experiment using a [DO-178B] compliant software development process by NASA confirms that no errors were identified in the developed software [Hayh]. [DO-178B] contains sufficient information for first time users to implement a compliant software process.

5.2 Software classification

In the embedded application, software classified at levels A, B and E has been realised. Partitioning software is produced to allow software of various level to run on the same processor. At the end of the project 81% of the modules are classified at level A, 8% at level B and 11% at level E. The increasing number of data fusion requirements lead to a larger share of level A software at the expense of level B software. With the small amount of level B modules remaining it is unclear whether the advantages of less rigorous testing of level B software outweigh the more complicated software development process.

When software classified at different levels has to run on the same processor, special partitioning software guarantees that software of one level can under no circumstance compromise the functioning of software at other levels. This partitioning software consumed only 1% of the total project effort. Even if all level B software would be developed as level A software, the partitioning software remains necessary and cost effective for separation of level A and level E (mainly maintenance) software.

5.3 C language

The C programming language contains numerous constructs that are unspecified, undefined or left to be defined by the compiler supplier [Hatt]. The C programming language is considered a project risk. This risk was reduced by choosing an ISO C-90 (also known as ANSI-C) compliant compiler complemented by a project coding standard defining, amongst others, a safe subset of C. Compliance to this project coding standard can be verified automatically by customising a commercial tool. The tool verification required by [DO-178B] revealed that the version management by the tool supplier turned out to be inadequate. The tool was already marketed world-wide since 1986 to hundreds of customers. This illustrates the rigour of the applied verification processes.

5.4 Requirements evolution

Due to the commercially defined short time-to-market, the customer defined the system requirements concurrently with the software requirement process. Before the start of the software design process the resulting analysis was subjected to a number of informal detailed technical assessments, performing the formal requirements verification activities with the exception of the certification authority involvement.

To aid the integration of the embedded application with the displays, co-developed by the customer, and subsequently with the avionics suite of the aircraft, a first version of the software with limited functionality was delivered before completion of the software requirements and software design processes. The first version served its purpose well. A lot of feed-back was obtained, resulting in many changes to and clarifications of the system requirements. Figure 1 depicts the resulting requirements evolution from the project start. Every point indicates a formal delivery of a working prototype or system to the customer. Figure 1 is cumulative: the number of partially implemented requirements is added to the the number of fully implemented requirements. Superimposed is the number of requirement changes for each delivery. The status of a requirement in a delivery can be:

- fully implemented;
- partially implemented i.e. the delivery only partially complies with the requirement and additional work is needed arrive at full compliance;
- not implemented, i.e. no part of the requirements is included in the delivery.

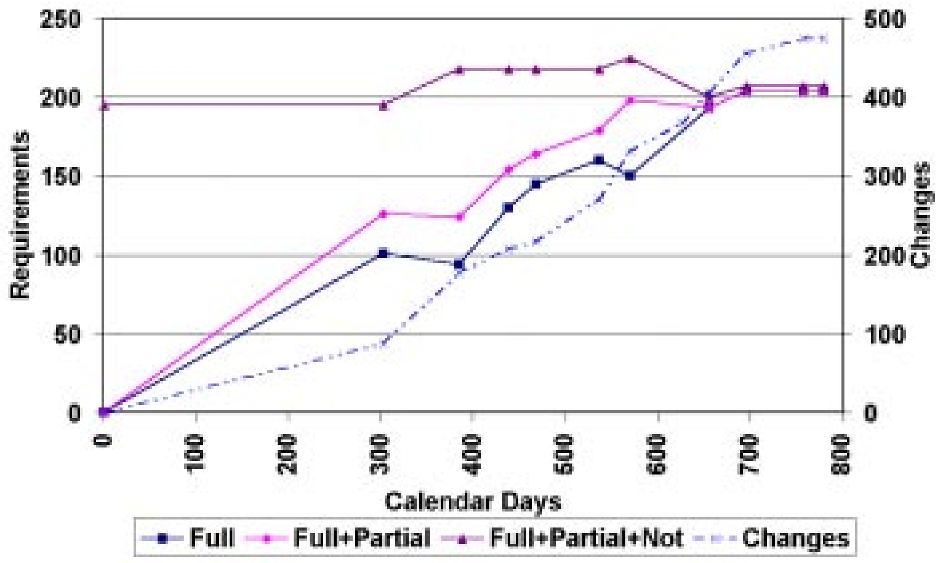


Fig. 1 Evolution of requirements and their implementation status.

The increase in the number of requirements and the reduction in the number of implemented requirements after 300 and 520 working days are caused by new issues of the requirements document.

The changes are caused by (in descending order):

- changes in the Human Machine Interfaces (HMI) of the aircraft. These changes originate from pilot comments and can only be obtained from demonstrating a working prototype in a realistic environment. Co-development of the displays and the embedded application helps to reduce the amount of changes on system level;

- adding product features. Apart from marketing input, these changes also result from experience with an actual prototype;

- integration of the embedded application with the displays and the aircraft subsystems. Formal methods to specify these interfaces might have helped to reduce this class of changes;

- ambiguities in the plain English specifications. Especially for HMI related features an unambiguous specification method which is intelligible for pilots, HMI experts and computer software experts is needed.

The requirements evolution combined with the need for intermediate versions resulted in a change from the waterfall model to the spiral model. For the non-certified versions the formal reviews were replaced by technical reviews with the same contents but without the external attendants. The multiple deliveries implied frequent integration with the avionics suite at the customer's site. This resulted in the combination of our team with the customer's display team on one site. Of the 15 deliveries only the ones at 655 and 779 calendar days have been certified. Note that the non-certified versions are not be used in flying aircraft.

5.5 Design evolution

Figure 2 shows the evolution of the number of modules (files containing C code) and external functions over time.

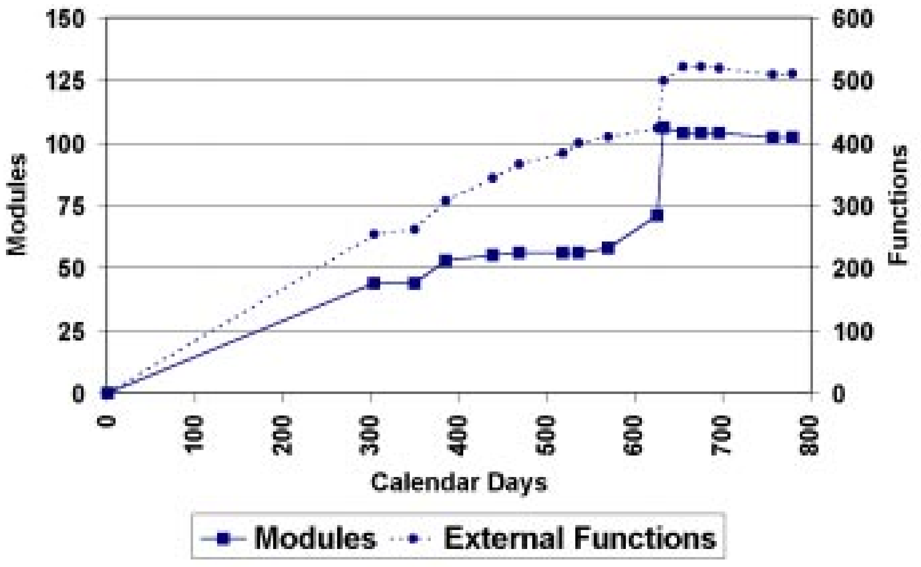


Fig. 2 Module evolution.

Up until the first certified version the number of modules increased only slightly, indicating that all changes could be accommodated in the original design. Due to different verification requirements, software of different levels was split into different modules for certified versions. The sharp rise in the number of commonly developed modules just before the first certified version is caused by this splitting. Evolving data fusion requirements influenced the safety classification of some functions. Some simplifications of a communication protocol for the second certified version resulted in a minor reduction in the number of modules.

The number of external functions rose approximately continuously until the first certified version, in accordance with the number of implemented requirements. The number of functions remained virtually constant for the second certified version. This indicates that the design remained relatively stable, most requirement changes could be accommodated in the existing modules.

On average there are 5 functions per module. On average each file has been submitted to configuration control 13 times. These changes are concentrated in one configuration item, the second configuration item became stable after the version of day 536. The remaining 2 configuration items remained unchanged after the version of day 438.

These results support the view that also in an environment with significant requirement evolution a sufficiently mature design is needed before starting the coding process. The design team leader ensured that the design remained uncompromised during the entire realisation period.

5. 6 Code size evolution

The code size evolution is shown in figure 3. Its continuous increase until the first certified version corresponds with the continuous increase in the number of implemented requirements. The subsequent slight reduction mirrors some requirements simplification.

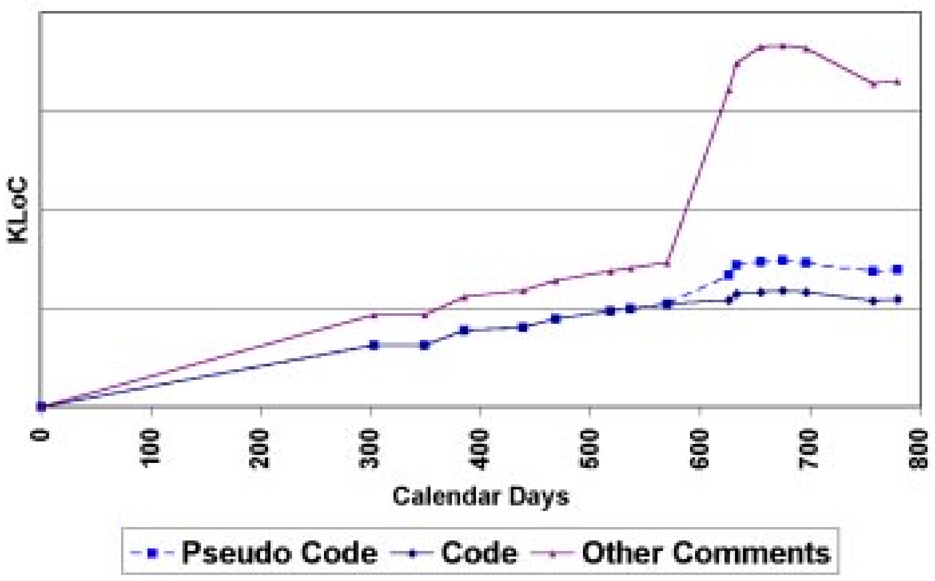


Fig. 3 Code size evolution

The CASE tool used only allows to progress once from analysis to design and once from design to code. It does not provide adequate support to incorporate analysis or design updates into the next phases. The amount of effort needed for data input even makes updating the analysis or design model cumbersome. After day 500 it was decided to retain the analysis model but limit its depth in order to allow for its updating.

The design model was abandoned as the CASE tool data input effort became unaffordable with the requirements evolution. Instead pseudo code was added to the code. The pseudo code contains an abstract description of the code in about 27% of its size. Also all interface definition information was added in extensive headers per function. This extra information explains the considerable increase in the amount of comment before the first certified version. The comment has a size of about 175% of the executable code.

On average each line of executable code has been modified 13.4 times, each comment line only 4.1 times. Changing the design information from the CASE tool to comment resulted in considerable manhour savings, at the expense of a transition period with a less intelligible design. The design team leader and the verification team leader had sufficient knowledge to answer any question on the spot. With a maximum team size of 16 people located on one site this turned out to be a workable solution. The changes break down in about 60% changed lines, 15% deleted lines and 25% added lines. As the product grew in size over time more lines were added then deleted.

5.7 Code breakdown

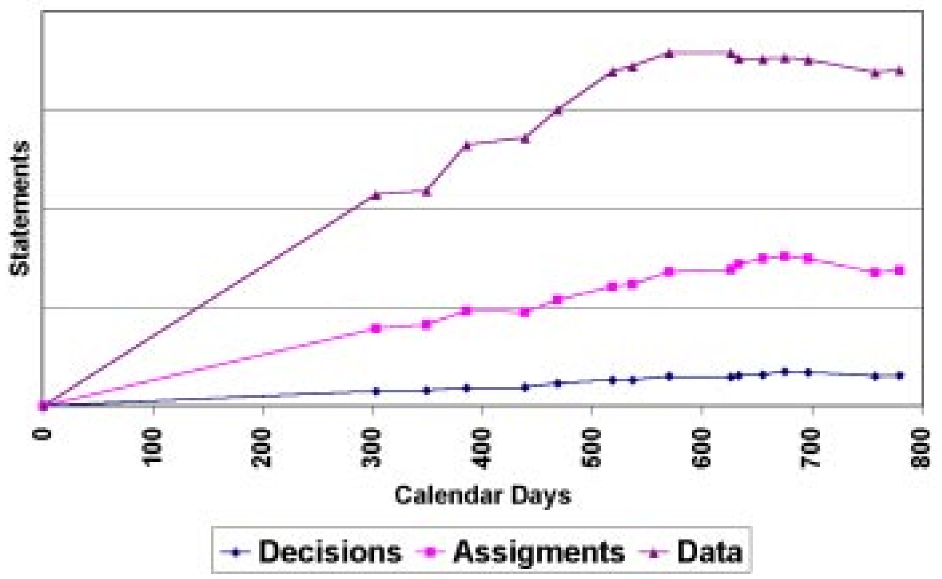


Fig. 4 Evolution of statement type distribution.

For testing purposes a division of statements is made into :

- decisions and loops (consisting of the "switch", "if", "for" and "while" statements);
- assignments;
- data e.g. tables.

The results are shown in figure 4. All statement types increase approximately continuously until the first certified version, with a slight decrease up till the second certified version. The system design was already based on maximum configuration possibilities using data files. Adapting the software behaviour to specific aircraft configurations by configuration files has the advantage of obviating re-certification. The real-time constraints caused some run-time optimised solutions. Experience with the various prototypes lead to more sophisticated solutions which comply with both the real-time requirements as well as with the requirements evolution. In the second certified version for each executable statement there is 1.48 lines of data. The statement type distribution refelects the design based on maximum use of data for configuring the software behaviour. The run-time optimisations are not reflected in a change of the statement type distribution.

5.8 Verification

Each testable requirement is identified to allow traceability from requirements through all development phases to verification. Every [DO-178B] compliant development phase contained full traceability of each requirement, by including the requirement identification. This has greatly helped the management of the virtually continuous requirement evolution. A lesson learned is to allocate a separate identification to each verifiable part of a requirement. [Hayh 1998] reached this conclusion independently.

A standard applies for the software requirement process. It's application has to be verified. Some simple tools can be produced to cost-effectively reduce the analysis effort. The same holds for the design standard.

For module tests the use of a Commercial-Of-The-Shelf (COTS) test tool greatly reduced the time needed to prepare the tests and to perform the regressions tests for each delivery. The actual test code is generated from Test Case Definition (TCD) files. On average each safety-critical function (i.e. [DO-178B] level A+B) is called 3.8 times during the verification tests. The non-comment part of the test case definition files equals 2.9 times the non-comment size of the code. The test comment grew to about 70% of the executable test case size implying that tool-assisted module testing still consumes a significant mount of effort. Due to the size of the test case definition files, comment is needed to document their function, to aid traceability, to improve readability, etc.

[DO-178B] requires data to be verified by inspection, only decisions and assignments can be verified by testing. For each testable statement 20 checks have been performed. For global data the test tool automatically checks that no global data is inadvertently changed, causing the large amount of checks per testable statement.

Integration testing was based on the white box approach. It comprised the correct functioning of combinations of functions. Integration tests also verified 19% of the requirements. These requirements could not be verified by black box testing only. Examples of the latter are spare processor time and spare memory requirements. During integration 184 tests have been performed. The COTS test tool did not support the multiple-module integration testing.

During validation testing the requirements are verified using a black box approach. Several requirements can be verified in one test. The 132 tests verified 90% of the requirements.

Analysis was used to verify 12% of the requirements. Note that some requirements can only be verified by a combination of analysis, validation testing and integration testing. Consequently the 3 percentages add up to more then 100%.

6. Conclusions

[DO-178B] compliant software processes have proven adequate for safety-critical software development.

Measuring the requirements evolution (refer figure 1) combined with the co-development need for intermediate versions resulted in the change from a waterfall software development process to a spiral software development process.

For a certifiable, safety-critical product with a commercially determined time-to-market co-development is a solution. The various prototypes, with increasing number of implemented requirements (refer figure 1), provided by a spiral software development process support this.

A sufficiently mature design is needed before starting the coding process for the first prototype. The design team leader has to ensure that the subsequent software modifications do not compromise the design. The module evolution (refer figure 2) needs to be complemented by other information to assess the design stability.

Metrics help in analysing and controlling the software processes. For example the evolution of requirements with their implementation status (refer figure 1) and the module evolution (refer figure 2), help in the trade-off between the date of the next delivery and its functions.

The CASE tool used did not adequately support design updates rendering it incompatible with the spiral model. Detailed design and interfaces can be included as comment in the code, to be automatically retrieved for the required documentation. The added source code (refer figure 3) turned out to be acceptable.

The statement type distribution (refer figure 4) refelects the maximum use of data to configure the software for each specific aircraft.

C combined with an appropriate coding standard and an automated analysis tool can be used for safety-critical certifiable software.

For some analysis tasks simple tools can be produced which cost-effectively reduce the analysis effort. The COTS test tool significantly reduced the testing effort.

References

[BS7925-2] British Standard software testing part 2: software components testing (august 1998)

[Dekk, Kess] Product Assurance For The Development Of The SAX AOCS Application Software, G.J. Dekker, E. Kesseler (1996) ESA SP-377, NLR TP-96167

[DO-178B] DO-178B, Software Considerations in Airborne Systems and Equipment Certification, (December 1992)

[DOD] DOD-STD-2167A Military Standard Defense System Software Development (1988)

[FAR/JAR-25] Federal Aviation Requirements/Joint Aviation Requirements FAR/JAR-25

[Hatl, Pirb] Strategies for real-time system pecification, Hatley, D.J., Pirbhai, A. (1988) Dorset House Publishing

[Hatt] Safer C, Hatton L., (1995) Mc Graw-Hill

[Hayh] Framework for small-scale experiments software engineering, K. J. Hayhurst

[IEC 61508] IEC 61508 Functional safety:safety related systems, 7 parts,   (June 1995)

[IEEE, 1998] IEEE, Developing software for safety- critical systems, J.Besnard, M. DeWalt, J. Voas, S. Keene (1998)

[ISO/DIS 15026] ISO/DIS 15026 Information technology - System and software integrity levels (1996)

[Kess, Slui] Safety and commercial realities in an avionics application, E. Kesseler, E. van de Sluis, Second World Congress on safety of transportation, NLR TP 97669 (1998)

[SAE ARP 4754] Society of Automotive Engineers Aerospace Recommended practise 4754, Certification considerations for highly-integrated or complex aircraft systems, (November 1996)

SESSION 6:

Experience Packaging and Transfer

Systematic Experience Transfer Three Case Studies From a Cognitive Point of View

Eva Wieser(1) , Frank Houdek(1,2), Kurt Schneider(1)
{eva.wieser, frank.houdek, kurt.schneider}@daimlerchrysler.com

(1) DaimlerChrysler AG
Research and Technology
P.O. Box 23 60
D-89013 Ulm, Germany

(2) University of Ulm
Computer Science Department
Software Engineering Group
D-89069 Ulm, Germany

Abstract

Individuals learn from experience no matter what they do. But what is natural for an individual is far less straightforward in groups or companies. There are some suggestions in literature how this hurdle can be overcome: The experience factory is a concept tailored for the software domain. In the tradition of this domain, however, the concepts are generally activity- or organization-focused and only rarely address cognitive issues.

At DaimlerChrysler, we were asked to establish experience transfer at the organizational levels in three business units. In three case studies, we saw a recurring pattern of cognitive tasks. While these tasks were carried out quite differently, there is a core to each of them that should not be neglected.

1. Introduction

In general, reuse is seen as a key to increasing quality or decreasing time-to-market or development costs [2, 21, 26]. The spectrum of reusable components can range from some lines of code and software architectures to project control metrics and complete development processes.

In particular, the idea of reusing *own* experience at the group or company level is fascinating, as it helps us avoid making the same mistakes over and over again. This kind of knowledge is related to the own environment, therefore adoption is less complicated due to the same constraints. Reusing experience means relying on insights rather than theoretical models or textbooks. In this context, we define experience as a 'collection of witnessings and insights gained by a human from the witnessings with respect to the world or to himself' [19]. Strictly, this implies that not experience itself (tuple of witnessing and insight) but experience knowledge (the insight) can be transferred. For the sake of simplicity, we use the term 'experience' instead of 'experience knowledge'.

Reusing experience in the own software engineering environment implies being able to capture experience in one project and to transfer and use it in another one. Since this activity usually exceeds the scope of the two projects, an additional organization is required to take care of it.

This idea founds the core concept of the experience factory approach proposed by Basili and co-workers [4]: Every time a new project (or activity) starts, processes, control metrics, products, etc. are selected from a collection of already-finished projects and tailored according to the new project's demands. After the new project has been finished, the gained experiences are added to the collection of experience (in the experience base). In this approach, a strong emphasis is put on the idea of measurement-based experience (e.g. error models, effort distribution models or quality models).

But from a cognitive point of view, this model makes some assumptions which do not necessarily hold true in practice: (1) all relevant experience can be collected, and (2) there is real need for experience knowledge, i.e. there are people willing to reuse it.

In our work at DaimlerChrysler AG, we observed experience collection and reuse in real projects where we learned to pay significant attention to the cognitive issues, i.e. how to transfer experience from a human-oriented point of view. In this paper, we analyze and reframe three projects as case studies for this cognitive task. By doing so, we demonstrate how varied experience transfer is.

The most important findings of our observation can be summarized as follows:

- There is a great variety of methods for learning and experience transfer. The measurement-based one is only one alternative among others.
- Experience transfer can happen by pull or push, i.e. it can be driven by concrete demands or by offering the available elements. In the second case, continuous encouragement and motivation is essential.
- At the beginning of a systematic experience transfer initiative, the role of external knowledge can become important for the achievement of first improvements.

1.1 Structure of this Paper

In Section 2, we shortly describe the organizational and cognitive frameworks used for experience transfer, the experience factory paradigm, and the cognitive experience transfer cycle. Section 3 presents our case studies on experience transfer. In Section 4, we discuss our observations and give conclusions for future activities. A discussion of our future steps (Section 6) end this paper.

2. Experience Transfer

Experience has always been seen as one of the assets of an organization. The software business is often characterized as depending on rapidly changing technologies and a high turnover in the workforce. Thus, an organization has to be able to learn from a small number of examples within a short time. This constellation requires a systematic approach to experience handling.

2.1 Experience Factory Approach

With the advent of the learning organization, growing attention has been drawn to the learning software organization [12, 17, 25]. The primary approach that was inspired by software engineering (rather than business or economics, as in [6, 27]) is the so-called experience factory (EF). The EF approach was first introduced by Basili at NASA-SEL [3]. Despite the fact that it gained new attention as an instantiation of a learning organization, the EF was initially a reply to the concept of a software factory [8]. Basili claimed that the software as such should not be the focus for reuse in a factory, but instead the experience behind the software.

From an *activity-oriented point of view*, the EF is basically a mechanism for the institutionalization of feedback loops. In publications, the *organizational view* of an EF is often emphasized [3, 4, 15] (see Figure 2). It shows a distinct EF unit facing several project units and a strategic planning unit. Whereas this clear distinction indicates EF independence of any project, arrows symbolize communication, interaction, and the back and forth of information. In several cases, we have seen how important it is to keep the balance between involvement and distance from project concerns [15].

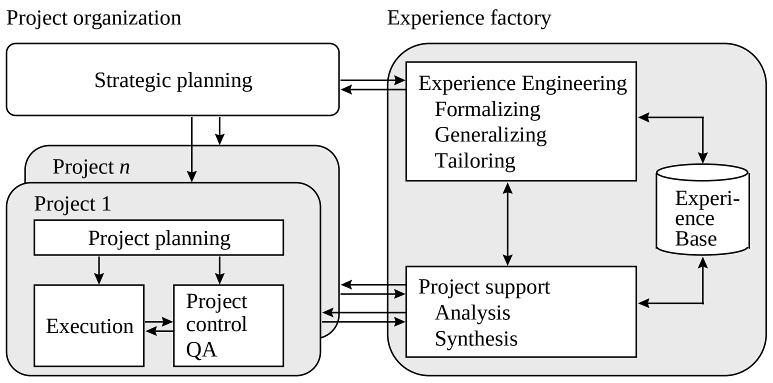


Figure 1: Experience Factory (EF)

The aspects of activity (what to do) and organization (where to do it) need to be taken into account when one establishes and runs an EF. Furthermore, we argue that it is essential to pay significantly more attention to the *cognitive view* of an EF (how to do it from a human-oriented point of view). In this respect, our model provides an additional dimension that we have found crucial in our EF projects. This dimension helps us to better understand what is going on around an EF, and it helps prepare us for aspects that could slip through the fingers in both organizational and activity views.

In a sense, we are most interested in the interplay of project organization and the experience factory organization when we recognize that this interaction is carried out by humans with their own interests, barriers, and motivations: What are the cognitive constraints on the arrows of Figure 2, and what needs to be done to keep experience and information flowing? We have seen in the case studies that a good answer to this question may be the most essential prerequisite for acquiring systematic learning from experience.

2.2 Cognitive Experience Transfer Cycle

We see the EF concept as a means to implementing systematic learning from experiences. Learning, however, is always influenced by cognitive factors [10, 30]. It takes place in a situational context that may dominate the importance of the contents to be learned: Depending on the learner's time pressure through work duties, for instance, and motivation in general, learning may face stiff challenges. Independent of content and the soundness of experience, poor motivation can be a reason for a stalled experience transfer. And motivation is nurtured more by the way people feel about information transfer than whether it is their task to carry it out or not. In other words, how and where experiences and derived advice is presented and the details of what is actually presented may be equally important. The same holds for eliciting experiences. When we neglected the cognitive dimension, we often failed to either solicit or to reuse experiences [15].

In order to cover the cognitive tasks, we adopted and adapted a model of organizational learning in the workplace [12]. Its origin lies in the field of Computer Supported Cooperative Work (CSCW). It has also been extended to bridge the gap between collaborative working and learning [20]. We agree that systematic learning shares several characteristics with group processes, so that the model can be applied analogously.

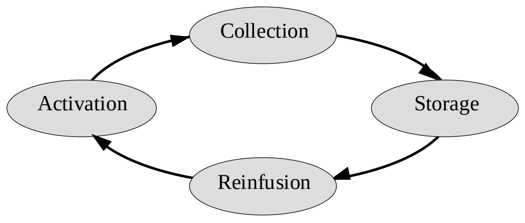


Figure 2: Cognitive Experience Transfer Cycle

Four tasks are placed in a cycle (see Figure 3). They all describe how experience (or, as in the original, design knowledge [12]) can flow from one place to another. All four tasks are crucial, and a poor job in carrying out only one of them can stall all of the other efforts, as well. This may seem obvious at the superficial level of task interpretation. However, when we take a slightly closer look, challenges and pitfalls become more interesting.

- *Activating experience:* This task deals with a problem that is known in the field knowledge engineering [16, 24]: Even people who have expert knowledge of a subject may be unable to convey their knowledge or experience. One typical reason is that people do not know what kind of information others need or what they consider to be experience. Another typical reason is that people often do not even know what they know. Polanyi calls this tacit knowledge [23]. Unconscious experiences of this kind need active help to become surfaced and voiced.
- *Collecting experience:* Depending on where and how experience is activated, there are different opportunities to capture it as it surfaces. In the easiest case, experience knowledge may be verbalized or written down by the person who has the experience. In more complex cases, experiences may be activated in daily work, but then there must be an easy way of capturing it [22, 25]. This requires some means of storing it, means of communicating it to the EF, and the motivation of the participating people to use both [29]. When activation of experience is planned, the chance to be there when it happens is also improved.
- *Processing and storing experience:* Theoretically, storage is not an issue. Everything from databases to the Internet is available to store data in. In practice, however, storing becomes a problem of prioritization and decision. Not everything can be stored electronically. Limited resources, and more especially limited motivation, force any EF to develop a pragmatic and feasible concept of what and how to document and store – completeness is inachievable [1].
- *Making information relevant to the task at hand (reinfusion):* This task is most often neglected. We found that many people consider it to be merely a technical problem of making results available. Nowadays, the Internet or Intranet seems to be the solution. From a cognitive perspective, however, pure delivery of results is far from sufficient [12]. Much more emphasis must be put on making this information helpful or relevant. An experience or experience derivative is rarely helpful in general. It only can be helpful for carrying out a certain task. Knowing what task needs to be worked on is, therefore, a prerequisite to collecting and processing gained results into something useful [17].

3. Case Studies

The case studies took place in real projects within Daimler-Chrysler. The topics which were the focal point of experience transfer were (1) defect detection, i.e. experience about the effectiveness of early defect detection techniques in the software lifecycle, (2) project management, i.e. experience about project control and project tracking, and (3) software contracts, i.e. experience about writing contracts for outsourcing software development activities.

3.1 Case Study ‘Defect Detection’

In this case study, we deal with measurement-based experience, which is most closely related to experience transfer as intended by the experience factory approach. Unlike the other two studies, the result of the activation-collection-storing activities are abstract models rather than concrete advice for software development.

3.1.1 Context

The observed project is concerned with the development of embedded software for automotive systems. Software is developed in increments, where each increment can be seen as a small project of its own.

Since these kinds of systems have to meet the highest quality demands, much effort is spent on defect detection activities. In this environment, the mission of the experience factory was to establish more efficient defect detection processes in order to reduce the effort required for rework on errors recognized too late in the process chain.

This mission was broken down into the following rough steps: Introducing software inspections for program code for all of the increments, assessing the quality of the inspections, and reusing the findings on the efficiency of the inspection in order to improve not just the inspection process but also the development process. To assess the quality of the inspections, we measured the inspection data (i.e. effort, preparation time, number of defects found, etc.) and the amount of errors found in field testing.

The experience factory implementation used here is, therefore, based on measurement programs.

3.1.2 Experience transfer

As is usual in measurement initiatives according to GQM (which was the selected approach, see [5]), we carried out interviews with the project leader and various developers in order to be able to identify those not so obvious facts to look for, such as particular error classes. By establishing the measurement program, e.g. forms or measurement processes, the people involved became more sensitive with respect to the topic observed.

The data on the defined metrics was collected both during inspections (e.g. effort, preparation time, number of defects) and in the later field-testing activities (e.g. number of defects and related modules). This data was validated through discussions with the developers involved and afterwards processed in charts and figures. Figure 4 depicts one of these charts. It shows the relationship between defect detection intensity and preparation time. The numbers inside the graph denote the sizes of the documents in terms of number of pages.[13]

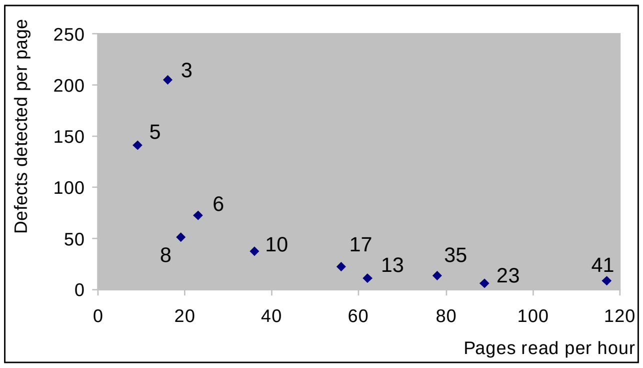


Figure 3: Relationship between preparation time (pages read per hour) and efficiency (defects found per page)

This figure illustrates that larger documents were read less intensively than smaller ones and, therefore, fewer defects were detected in these documents. The main reason for this was the enormous time pressure which made it impossible for the inspectors to spend more time on their preparation.

Another finding which was gained by measuring testing activities is depicted in Figure 4. This graph shows that more field-errors were reported in larger modules than in smaller ones. A more detailed analysis (i.e. comparing inspection intensity and field-errors for each module) confirmed this trend.

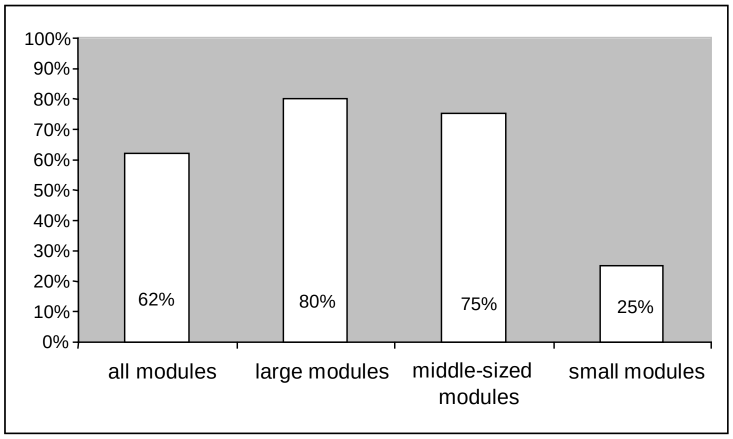


Figure 4: Distribution of field-errors with respect to module size

The experience gained in inspection intensity and its influence on field-errors was reported to the project manager. On the basis of these findings, he decided to expend one additional week of effort solely for inspection activities.

Further measurement will show whether the proposed hypothesis ‘more preparation time will reduce the number of field-errors’ will hold or not.

---

[13] The real numbers are re--scaled to protect company internal information.

3.1.3 Cognitive perspective

As the underlying mechanism of experience collection used here (QIP [4]) is well defined, the cognitive tasks were also performed in an orderly fashion. Activating, collecting, and processing the experience correlated with goal identification, data collection, and data analysis, respectively. But there is no one-to-one correspondence. Goal and metric identification (primarily activation) has a great deal to do with collection (e.g. deployment of abstraction sheets [9]) and storage (e.g. documentation of the results of the interviews and discussions).

The outcome of this measurement-based experience collection was a quantitative model rather than concrete advice upon a specific problem. This makes the reinfusion step much harder since there is no task at hand for which the created experience package is immediately relevant. It is the duty of the experience manager to advertise the findings hoping that someone will be interested in them (in our case, this was the project manager, who was willing to reuse the experience by modifying his process). This makes the risk of ‘push’ obvious; the produced experience packages might not be reused.

3.2 Case Study ‘Project Management’

In this example of experience reuse, the knowledge of how to track projects was transfered. It started with external knowledge infusion and was initiated by a ‘pull’ effect, a request for information. Later, ‘push’ actions dominated, as the detailed tracking of projects must be carried out continuously and thoroughly.

3.2.1 Context

The experience reused in this case is about how to plan and track projects so that the current state of work is well recognized at any time, including any reasons for delays. Process improvement activities served as the source of the experience gained. They were carried out beforehand and were continued interweavingly. In the projects, highly complex embedded real-time systems were developed that range from middle-sized to large.

3.2.2 Improvement activities

The results of an extensive characterization of the organization identified project planning as a primary field for improvement. Methods of project planning in literature were examined and tailored to the concrete needs. First results were a better planning template and several metrics to track project progress and planned vs. unplanned activities. For the data analysis, two procedures were written that allowed automatic evaluation. Feedback on the measurement program lead to minor changes and enhancements of the evaluation procedures.

3.2.3 Creating experience packages

Processing the gathered information resulted in several experience packages describing the best practice of project planning and tracking as well as effort estimation and tracking. Their basic structure is depicted in Figure 5.

The packages (gray shaded boxes) were put into HTML pages following the Quality Pattern structure [14], which, together, form a Quality Pattern System [18]. The white boxes sketch attached, but not further processed documents.

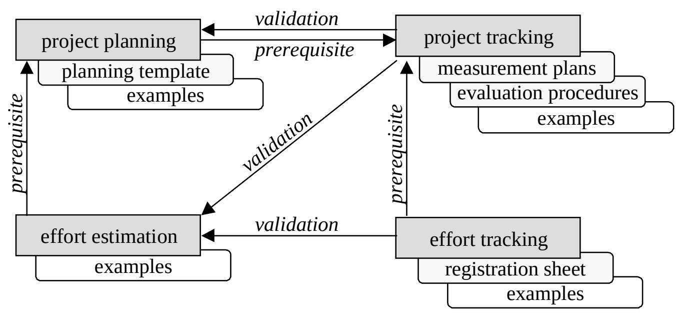


Figure 5: Structure of experience package system

The experience packages were reused in another project for which planning also turned out to offer a chance for improvement. The first exchange was carried out through an interview where both the leader of the old and the leader of the new project took part. They exchanged information about their projects, discussed commonalties and differences.

The meeting served as a starting point for active project support which the leader of the old project rendered to the new one. In our role as mediators, we had the chance to capture characterizations of the projects that would serve as a basis for future tailorings, i.e. how to deal with different project lengths, different phases in the development process, different strategies in changing requirements, different test strategies, etc.

The improvement of project management showed its success. The continuous analysis of the project led to process improvement actions that resulted in the project being on time.

3.2.5 Cognitive perspective

The experience cycle started with the step ‘make experience relevant to the task at hand’ based on the need for information that can be characterized as a ‘pull’ effect. External experience, i.e. literature, had to be made relevant by adjustment.

Despite the automated evaluation procedures, the awareness that tracking projects better helps to keep the schedule (as you are able to recognize delays earlier) and to build a profound basis for future estimations and planning had to be upheld through many discussions and continuous training, which indicates a clear ‘push’ situation.

Without personal coaching, the transfer between the two projects might easily have failed: the access to the information and a one-time explanation are simply not enough for a successful transfer, as some earlier efforts within the first project proved. A second advantage of personal coaching was given by the fact, that the information to be exchanged cannot always be prepared in time, despite the usefulness of a nice form.

Transferring the information about project planning activated implicit knowledge, as the project leaders, upon their turn to talk about project planning, came to exchange tips and tricks about how to lead projects. We also found the acceptance of our initiative on planning and tracking quite dependent on the personality of the people concerned. If the mission contradicts the interests of someone, she will not carry out the mandatory tasks thoroughly enough.

3.3 Case Study ‘Contracts’

In this case study, quotations and tips in contract design were exchanged. We encountered a ‘pull’ situation in which experience is requested for reuse before it has been collected and stored. The qualitative techniques deployed show quite different perspectives on the cognitive tasks as in the first two cases.

3.3.1 Context

Software development is not considered a core competence in all of the DaimlerChrysler business units. As a consequence, software development is often out-sourced, leaving DaimlerChrysler with the task of carrying out acceptance tests. When the experience factory started in this part of the company, a one-day workshop was devoted to finding goals and defining topics of interest. Acceptance processes were selected as the primary target. This area was identified as one in which there was room for improvement.

We started to activate experiences about acceptance processes by a series of interviews in order to gain a better understanding for the concrete problems and possible solutions that had been tried. For this purpose, open interviews seemed most appropriate. An interviewer was equipped with about a dozen rather general questions. During the interview, the respondent was encouraged to answer beyond the strict limits of the questions put. No sequence was imposed, and the interview should flow and be more like a conversation.

Another technique for opening up a business unit for experience transfer, a half-day workshop, was also implemented. People who had been carrying out actual acceptance processes were brought together to discuss questions on these issues (some of which had been raised in the interviews).

3.3.2 Experience transfer

Two of the people involved in the interviews or the workshop mentioned contracts and their influence on their acceptance efforts. One respondent reported a negative experience during acceptance due to a bad contract, while the other was proud to be able to tell that some contractual agreements helped them tremendously during acceptance testing. Those statements were documented but not immediately pursued. The main topic of interest was still the late phase of acceptance testing in the narrow sense.

Some weeks later, there was a request by someone else in the same business unit who had heard about the experience factory. Being in the phase just before signing a contract, he felt uneasy and wanted to make sure there were no known flaws in the draft contract. Even though all contracts are thoroughly checked by lawyers and business people, software engineering concerns had sometimes been neglected in the past. When this person asked the experience factory for assistance, he found that there had so far not been much experience or knowledge collected on the subject.

Nevertheless, this situation seemed to provide a unique opportunity: to demonstrate how the early phases (such as requirements and contracts) determined the success of late phases such as acceptance testing. Therefore, we started with immediate research on the topic, and the following steps were taken within fewer than two weeks as the contract had to be signed by a certain date:

1. Review interview protocols in which contracts were mentioned.
2. Carry out follow-up interviews with these people that were now focused on contractual issues.
3. Copy interesting parts of contracts and agreements that had been referenced in the interviews.

4. Check company standards and literature.
5. Carefully read through draft contract and comment on it.
6. Summarize three top recommendations, including a passage taken from one of the other contracts. We urged the problem owner to consider these issues even if there should not be enough time left to work through all of the other findings.
7. Institute a short follow-up meeting with the problem owner and discuss what we consider important and why.
8. After a few months, the problem owner reported on the project again, supporting many of our recommendations.

This process was not ideal, and we do not claim complete results for this kind of ad-hoc research. Some interesting results were achieved:

- The result helped where and when it was needed.
- The topic was recognized as an important area for experiences that have since then grown.
- We encountered several analogous situations in which a marginal topic was addressed and had to be dealt with within a few weeks.

Consequently, these kinds of situations must be taken seriously from a pragmatic point of view.

3.3.3 Impacts

Contractual agreements have since been turned into one of the most active experience exchange issues in that experience factory. It could start from a reasonable seed [10] of material:

- the response to our advice as well as some later feedback,
- a neutralized experience package that dealt with the most important questions raised during the interviews,
- a list of known pitfalls, such as blind reliance on ISO 9000 certification.

Several projects have received the experience package in the meantime. There was generally a personal meeting afterwards to explain the experiences and gather new input.

3.3.4 Cognitive perspective

The topic area of contract review was not approached using measurement. The experience on this topic was neither intentionally pursued nor *activated*, but an anchor remained in the interviewers' minds. Only later was there a need to reuse experiences that someone believed we had collected. How to make the findings *relevant to the task at hand* was not a concern: the task was perfectly clear, and the request ('pull') proved that there would be no motivational problems. The cognitive experience transfer cycle started at the *make relevant...* part which then triggered one fast turn of the cycle:

- fast activation with focused interviews
- straightforward collection in interview protocols and written recommendations
- storing the data was not a high priority. Almost immediately this was tailored to be delivered. Intermediate storage happened in brains and on scrap paper. Afterwards, the findings and results were neutralized and documented.

There is always the temptation to quickly satisfy pull requests but to never document them. If this happens, one operates in the (usually well-known) fire-fighter mode. It is mandatory for any unit that wants to learn from experiences systematically to solicit feedback on what the advice did and then to go through all of the cognitive steps – even if it happens after the pull activities.

Cogn. task	***Defects***	***Project management***	***Contracts***
Activate	GQM goal identification; Interviews, abstraction sheets; Feedback	Interviews; GQM goal identification; External knowledge acquisition	By chance: mentioned in different context; Later by focused interviews
Collect	Measurement; Protocols; Feedback meetings	Measurement program documents; Meetings with involved people	By interviewers, using question-lists; Copying contracts
Process and store	Statistical data analysis; Feedback with participants; Store in database	Store in a web-base; Presentation and feedback sessions with all potential users	Excerpt, compare from contracts; Neutralize and abstract; Write three-page sheet with hints and 'recommended wording'
Make relevant, reinfuse	Presentation for project manager in his planning phase	Meeting for direct exchange; Personal coaching	Simple, since reuse was requested (pull): phone call or short meeting

Table 1: Implementation of the cognitive tasks in the case studies.

4. Conclusions and Discussion

Learning from experience at the group or company level is a complex and multi-faceted task. This is true both for activity- and organization-based issues as well as for cognitive ones. Table 1 summarizes our activities in the three projects with respect to the cognitive experience transfer cycle.

Beyond the differences in the concrete actions (e.g. holding a feedback session or copying contracts), there are also superficial differences. In the defect detection study, the situation can be characterized as a push situation, where the‘contract’ study shows a clear pull situation. The ‘project management’ study shows a combination of both.

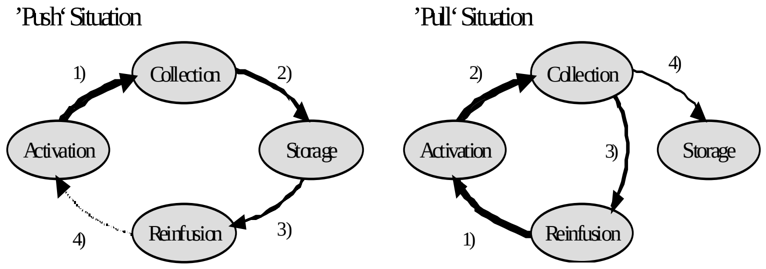


Figure 6: Instantiation of the cognitive cycle in push and pull situations

Another major difference is the source of experience. In theory, when following the concept of reusing (own) experience, external knowledge might be ignored, since it is not derived from witnessings in the own environment. In practice, there will be always a combination of internal and external knowledge. In the project management study, external knowledge was used for first improvements.

The two characteristics (pull/push, internal/external) have a great impact on the cognitive tasks, as well. Figure 6 illustrates this graphically for the pull/push characteristic. There, the numbers denotes the order of tasks and the width of the arrows indicates intensity and clarity assigned with the corresponding tasks (e.g. in an pull situation, it is clear how to reinfuse the gained experience, whereas in a push situation this task is most uncertain).

There are some implications of our cognition-based observations with respect to other experience transfer activities:

- As a rule, experience is not available as is. It has to be activated (tacit knowledge) or even exposed (e.g. by measurement activities). In particular, collecting data is not enough. Rather, measurement data is little more than the trigger for insight into deeper software developing issues, and this insight is the actual purpose an outcome of measurement.

- The delivery of experience packages is a necessary prerequisite for reuse, but it alone is not sufficient. Experience packages must be offered at the right time for the right task in the right form.
- Reuse depends not only on organizational issues but also on the attitude of the people involved. In particular in push situations, you have to motivate and encourage them until they are with you.
- The presented cognitive experience transfer tasks (Figure 3) describe a mental model rather than a fixed sequence of steps. Their concrete implementation (both at the individual level and across different activities) can look very different.

Especially in pull situations there is always the temptation to neglect the storage task, as this task competes with reinfusion activities as, for example, making presentations. For sure, providing automated support for processing and storing lowers this temptation.

5. Future Work

We have shown three different case studies that shed light on the cognitive dimension of running an experience factory. Given that activities like the three cases described in this paper are successful, there will be a steadily growing flow of experiences. Not all of those experience items, however, will stay valuable forever. Some will become outdated, others will contradict each other. There the management of experiences becomes a crucial topic. Fischer et al. [13] have treated this problem and presented their so-called Seeding-Evolutionary Growth-Reseeding Model (SER model). So far, our cases are all busy ’seeding’ experience collections. We are currently reaching ’evolutionary growth’. Once a critical mass has accumulated, we will focus more on issues of experience ’reseeding’.

The cognitive experience transfer cycle presented in Figure 3 and used as a reference throughout the rest of the paper also has implications for our future work. Among other things, such as building decision tables to select appropriate activation strategies in different situations, we are interested in tool support.

As a rule, an EF will have an electronic storing component. Up to now, we have experimented with the Intranet as an ubiquitous medium for information exchange. Not surprisingly, one single way of putting in experience is seldom an adequate way of activating experience. As the case studies have shown, different approaches from measurement to interviews to workshops can be applied. When a tool is involved, there may be even more options [7, 11, 25, 29]. Fischer et al. [10] talk about ‘reflection-in-action’ as the mode in which a knowledge-worker can reach previously tacit knowledge. The same is true for experience. Fischer argues that we need to create ‘breakdowns’ intentionally to stimulate such a reflection act [10]. Either a person or a tool has to activate knowledge: Both will use approaches that are most suitable for their kind of contact to the problem owner.

Along the same line, the role of a tool in experience distribution is far more ambitious than a mere database query. The challenge is to say the right thing at the right time in the right way [28]. When we try to embed experience delivery into a tool that is used to carry out the task at hand, we have a much better chance of making it relevant to that task.

The above observations will shape the tools that we wish to develop in the future: tools that reach from interview experience collection forms to sophisticated Intranet components. Analyzing the cognitive aspect of systematic learning from experience has reminded us of the final goal: better helping man to avoid making the same mistakes over and over again!

6. Acknowledgements

The authors wish to thank our colleagues involved in the EF initiatives, especially Heike Frank. Their contribution is gratefully acknowledged. Comments by Stefanie Lindstaedt helped to improve this paper.

References

1. Ackermann, M.S. *Augmenting the organizational memory: A field study of Answer Garden*. In Proc. Conf. on CSCW, Chapel Hill, 1994.

2. Basili, V.R. and Rombach, H.D. *Support for comprehensive reuse*. Software Engineering Journal, pp. 303-316, 1991.

3. Basili, V.R., Caldiera, G., McGarry, F., Pajersky, R., Page, G., and Waligora, S. *The software engineering laboratory – An operational software experience factory*. In Proc. 14th Int. Conf. on Soft. Eng. (ICSE'92), pp. 370-381, 1992.

4. Basili, V.R., Caldiera, G., and Rombach, H.D. *Experience factory*. In: Marciniak, J. (ed.): *Encyclopedia of Software Engineering, vol. 1*. John Wiley & Sons, New York, pp. 469-476, 1994.

5. Basili, V.R., Caldiera, G, and Rombach, H.D. *Goal question metric paradigm*. In: Marciniak, J. (ed.): *Encyclopedia of Software Engineering, vol. 1*. John Wiley & Sons, New York, pp. 528-532, 1994.

6. Brown, J.S. and Duguid, P. *Organizational learning and communities-of-practice: Toward a unified view of working, learning, and innovation*. In Organization Science 2 (1), pp. 40-57, 1991.

7. Conklin, P. and Begeman, M. *gIBIS: A hypertext tool for exploratory policy discussion*. In Trans. of Office Information Systems, 6 (4), pp. 303-331, 1988.

8. Cusumano, M. Japan's software factories: *A challenge to U.S. management*. Oxford Univ. Press, New York, 1991.

9. Differding, C., Hoisl, B, and Lott, C.M. *Technology package for the goal question metric paradigm*. Tech. report 281-96, University of Kaiserslautern, 1996.

10. Fischer, G. *Turning breakdowns into opportunities for creativity*. In: Special Issue on Creativity and Cognition, Knowledge-Based Systems 7 (4), pp. 221-232, 1994.

11. Fischer, G., Henninger, S.R., and Redmiles, D.F. *Cognitive tools for locating and comprehending software objects for reuse*. In 13th Int. Conf. on Soft. Eng. (ICSE'13), pp. 318-328, 1991.

12. Fischer, G., Lindstaedt, S., Ostwald, J., Schneider, K., and Smith, J. *Informing system design through organizational learning*. In Proc. Int. Conf. on Learning Sciences (ICLS'96), pp. 52-59, 1996.

13. Fischer, G., *Seeding, Evolutionary Growth and Reseeding: Constructing, Capturing and Evolving Knowledge in Domain-Oriented Design Environments.* In Automated Software Engineering Journal, Vol. 5, No.4, October 1998, pp. 447-464.

14. Houdek, F. and Kempter, H. Quality patterns – An approach to packaging software engineering experience. Soft. Eng. Notes, 22 (3), pp. 81-88, 1997.

15. Houdek, F., Schneider, K., and Wieser, E. *Establishing experience factories at Daimler-Benz: an experience report*. In Proc. 20th Int. Conf. on Soft. Eng. (ICSE'98), pp. 443-447, 1998.

16. Kidd, A.L. (ed.): *Knowledge Acquisition for Expert Systems*. Plenum Press, New York, 1987.

17. Landes, D. and Schneider, K. *Systematic analysis and use of experiences from software projects at Daimler-Benz*. In Oberweis, A. and Sneed, H.M. (eds.): *Software Management '97*. Teubner, Stuttgart, pp. 63-73, 1997 (in German).

18. Landes, D., Schneider, K., and Houdek, F. *Organizational Learning and experience documentation in industrial software projects*. In Proc. Workshop on Building, Maintaining, and Using Organizational Memories (OM'98), 1998.

19. Lexikon-Institut Bertelsmann, *Dictionary*, Bertelsmann, Gütersloh, 1979 (German).

20. Lindstaedt, S. *Group memories: A knowledge medium for communities of interest*, Ph.D. Diss., University of Colorado, Boulder, 1998.

21. McClure, C. *Extending the software process to include reuse*. Tutorial at the 1997 Symposium on Software Reusability, (SSR'97), 1997.

22. Ostwald, J. *The evolving artifact approach: Knowledge construction in collaborative software development.* Ph.D. Diss., Univ. of Colorado, Boulder, 1995.

23. Polanyi, M. *The tacit dimension*. Doubleday, Garden City, New York, 1966.

24. Puppe, F. *Systematic introduction to expert systems: Knowledge representation and problem-solving methods*. Springer, Heidelberg, 1993.

25. Schneider, K. *Prototypes as assets, not toys. Why and how to extract knowledge from prototypes*. In Proc. 18th Int. Conf. on Soft. Eng. (ICSE'96), pp. 522-531, 1996.

26. Software Productivity Consortium Services Corp. *Reuse adoption guidebook*. 1993.

27. Senge, P. *The fifth discipline - The art & practice of the learning organization.* Random House, London, 1990.

28. Stolze, M. *Visual critiquing in domain oriented design environments: Showing the right thing at the right place*. In Gero, J.S. and Sudweeks, F. (eds.): *Artificial Intelligence in Design'94*; Kluwer Academic Publishers, pp. 467-482, 1994.

29. Terveen, L.G., Selfridge, P.G., and Long, M.D. *From folklore to living design memory – Human factors in computing systems*. In Proc. INTERCHI'93, pp. 15-22, 1993.

An Experience Report on Decoding, Monitoring, and Controlling the Software Process

Luigi Benedicenti
Faculty of Engineering
University of Regina
3737 Wascana Parkway
Regina, SK, Canada S4S0A2
(306) 585-4701
Luigi.Benedicenti@dist.unige.it

Stefano De Panfilis
EngineeringIngegneria Informatica S.p.A
Via dei Mille, 56
Roma
00100, Italy
+39-6-492011
depa@mail.eng.it

Giancarlo Succi
Department of Electrical &
Computer Engineering
The University of Calgary
2500 University Dr. N.W.
Calgary, AB, Canada
+1 403 220 8357
Giancarlo.Succi@enel.ucalgary.ca

Tullio Vernazza
DIST – Università di Genova
Via Opera Pia 13
16145 Genova
Italy
+39-10-3532173
Tullio.Vernazza@dist.unige.it

Abstract

This paper reports on the experience in modeling the software process of a major business software producer located in Italy. The experience was conducted over a period of sixteen months, and was made possible by the European Community ESSI program (PIE no. 23699, DECO'). The modeling technique employed is object oriented coupled with activity based costing for process accounting. The baseline project used to model the process is the development of the information system for a large Italian municipality. The approach is innovative in that it empowers the actors of the process as process monitors, in addition to increasing their process awareness and understanding. Moreover, it tackles the problems of a distributed organization, which involve not only internal synchronization issues, but also the lack of precise communication rules with the customer. The main results are three. Decoding the process gave developers and managers complete visibility of the activities in the project: this identified the communication problems with the customer. Monitoring the process allowed profiling the activities, improving them especially in terms of role interchangeability. Controlling the process was therefore possible in a unified way, where side effects of any control action become immediately visible.

Keywords

Software Process, Modeling, Control, Experience Report

1. Introduction

This paper presents our experience in modeling the software development process of a major business software producer located in Italy: Engineering Ingegneria Informatica S.p.A.. While North American companies look favorably at innovations coming from both Business Process Modeling and Reengineering, European companies are not so willing to take the risks of a reengineering action. Therefore, the modeling effort described here acquires additional importance.

This experience was made possible for the following reasons:

- The software company is enjoying a period of good health, being ahead of the market and large enough to allocate the resources needed.

- The European Union is helping process improvement efforts in companies through special grants. This Process Improvement Experiment has been awarded such a grant (PIE no. 23669, DECO’).

- The Software Production Engineering Laboratory at the University of Genova, in cooperation with the University of Calgary, has produced a methodology for process modeling and reengineering.

2. State of the art

The software development process is receiving more and more importance, both from a theoretical and from a practical point of view.

Leon Osterweil [15][16] was one of the first to detect and point out similarities between software development processes and software itself. Since then, two main approaches have been consistently followed to deal with process modeling: a formal approach and an empirical approach.

The formal approach is based on very strong model semantics. Semantic clarity is indeed very useful for precisely defining a modeling base. Examples of this thread are the models based on Petri Nets [4][5][6], system dynamics [1][2], data flow diagrams [14], and state diagrams [10].

Formal models are perfect in all the situations where a formal development process is already in place and rigorous development methods are followed. However, when no such formal method is in place, they often lack the capability to cope with the sudden twists and variations that an informal business process may take.

In 1992, Curtis *et al.* [9] identified a series of objectives for software process modeling. They were as follows:

- Communication
- Improvement
- Management
- Automatic guidance
- Automatic execution

The objectives can be met provided that the process modeling method meets the following requirements:

- Shareable common model
- Reference basis
- Common process definitions
- Automated process handling tools
- Automated process enacting tools

Formal methods account for these requirements. However, there is still a general problem: the *hermeneutics problem* [3]. The hermeneutics problem consists of the impossibility to couple the formal semantics with meaningful extensions to account for variability introduced by the modeler's decisions.

Empirical methods often allow unclear semantics. Jacobson [12], for example, proposes a framework under which the modeler acquires control over the modeling process and semantics. Aksit [3] proposes a hermeneutic method that accounts for unclear semantics, adding semantically significant interpretations. Any addition is part of the model and therefore changes its semantics in a dynamic fashion. This solves two problems:

- The quantization problem
- The contextual bias

The quantization problem arises when the modeler operates a binary decision: if there is not the possibility to record the decision and backtrack from it, then the model is too rigid and the modeler may incur in quantization errors.

The contextual bias is introduced by the influence of the context on a certain decision. This influence is not easy to quantify.

Many empirical modeling methodologies rely on object orientation. Object orientation provides a framework on which it is easy to build. However, ill-conceived methods may lead to misunderstandings and model failure. The first empirical modeling methods were only loosely based on formal techniques. Ivar Jacobson [12] has presented a much more reliable method. His object oriented process reengineering is firmly set in an object oriented framework. However, his conceptual framework lacks a template object taxonomy. The knowledge within each model is tied to the modeler’s experience and point of view. Thus, replication of the model is possible only if performed by its creator.

Recently, the focus of the modeling and reengineering efforts has shifted to the cultural issues, that play a central role in every successful reengineering action [18].

Thus, the main qualities searched in a process modeling method are:

- Simplicity
- Independence from the modeler
- Validation

We believe that the modeling technique we developed has these qualities.

3. Modeling the process

3.1 Context

The company has recently acquired a large project for the development of significant parts of the information system supporting a large Italian Municipality. This information system aims at fulfilling the new and increasing responsibilities of Italian local administrations. It has a lot of characteristics that make it particularly suitable for the proposed experimentation:

- The project is divided into four interrelated subprojects, devoted to different subsystems: protocol, taxation, commerce, transparency (of administrative acts and procedures to citizens). This guarantees the temporal alignment of the experiment with the baseline application without introducing delays.

- The project foresees both the customization of corporate packages and the development from scratch. This requires the joint work of the Production Centre of Roma and the Package Department, located in Padova and Bologna; to some extent, the distributed nature of the company is under test.

- The contract imposes cogent quality, time and cost constraints that require deep levels of technical and economic control and good confidence in assuring quality.

The project is a good sample of the kind of projects the firm is normally involved in, because of its size, its distributed structure, its constraints, and its scope; this makes easy the internal transferability of the experience. The current wealth of the company is such that the project does not pose any particular problem, and thus there is no critical point in it. The project is structured in two different teams:

- the first, located in Roma, is in charge of the customer interfaces, system integration, "ad hoc" software development, installation and data conversion, system test and user acceptance;

- the second, located in Padova and Bologna, is in charge of the customization of the involved packages on the basis of the project needs. The first is a package for the complete management of city taxes, monitoring of evasions, and revenue planning. The second is a platform for the integration of information systems in managing administrative procedures. This allows simultaneous work on various aspects of the same dossier, gives management complete visibility of the process, and offers the citizen the means to examine and check how his or her dossier is proceeding.

Basically the Production Centre of Roma plays the client role with respect to the Package Department. Indeed, not only it presents to the Package Department the end-user requirements, but also the specific quality and productivity requirements.

A global Project Manager is in charge for the correct co-ordination and co-operation of the two teams with a specific care of their respective roles. The subprojects are planned to perform in parallel, and so it is. The involved staff includes:

- 1 Technical Manager
- 4 Project Managers (2 in Roma, and 2 in Padova/Bologna)
- 4 Analysts (2 in Roma, and 2 in Padova/Bologna)
- 8 Programmers (2 in Roma, and 6 in Padova/Bologna).

The part of the project modeled started in June 97 and ended in June 98, with a total effort of more than 10 man-years.

3.2 Methodology

The methodology was developed by the University of Genova in cooperation with the University of Calgary, and consists of the integration and enhancement of two existing techniques: Object Oriented Business Modeling and Activity-Based Costing [11][12][17]. The former technique employs object orientation to model the business process. The latter is an accounting technique that keeps track of the resources spent in cross-functional activities (Figure 1).

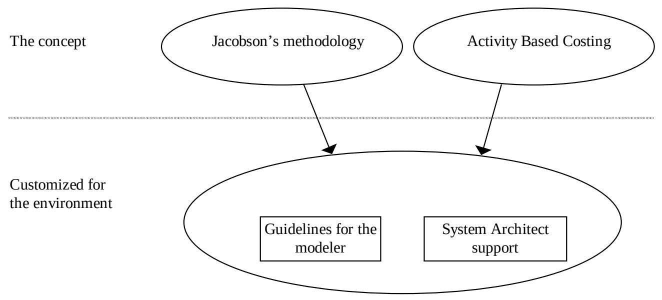


Figure 6: Development of the methodology

A full description of the methodology is in [8]. The details of the adaptation are not presented here, since they are a critical asset for the company.

3.3 Endeavor

The object oriented model was built as follows:

- **Identification of use cases -** A use case is a textual description of a process. Its shows the dynamics of a process. The modeler interviews the firm's employees to obtain the required information.
- **Identification of people -** This phase identifies an explicit set of people. The modeler examines the use cases and records every mentioned person.
- **Construction of People Interaction Diagrams (PIDs) -** PIDs provide a graphical representation of use cases from the perspective of people involved and messages exchanged.
- **Identification of roles -** People are related to activities by means of roles. Activities can work even if people are interchanged to play different roles in different times, as long as one person is assigned to each role an activity requires at any given time. The roles the modeler identifies must fit the use cases.
- **Identification of activities -** The modeler extracts the activities from the use cases. An activity is a set of operations performed to achieve a goal. An activity may require the cooperation of many people. Activities are atomic processes. Atomic processes can not be decomposed in simpler activities. Albeit simple, atomic processes must not be trivial. Activities are classified in three categories: Interface, Control, and Execution
- **Construction of Activities Roles People (ARP) snapshots -** The ARP snapshot grasps in a single diagram the interactions occurring in a use case. An ARP snapshot establishes links between people, roles, and activities.
- **Construction of Activities Interaction Diagrams (AIDs) -** AIDs provide a graphical representation of use cases from the perspective of activities involved and messages exchanged.

- **Construction of Activities Roles People (ARP) diagrams -** An ARP diagram focuses on the description of an activity as far as roles and people are concerned. The modeler constructs an ARP diagram for each identified activity.

Throughout all the modeling stages, modelers were supported by a drawing and reporting tool customized for the modeling environment.

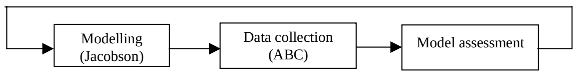


Figure 2: Interactive model building

The model was built in iterations following the scheme in Figure 2. The model was stable after 3 iterations. Additional information can be found in [7].

4. Results

The project yielded three results:

1. Process visibility. The process model refined with the coming of activity based costing data succeeded in giving managers and developers a uniform view of the process, allowing informed discussion and improvement strategies: “Decoding the process”.

2. The individual and group activity profiles: “Monitoring the process”.

3. Knowledgeable and informed project management: “Controlling the process”

The adaptation of the methodology to the firm analyzed is useful to the modelers, that have gained precious real-life experience on the field, but is rather specific, and shall not be discussed here.

4.1 Decoding the process

The process model depicting the interactions among the different activities and roles presented a clear picture of the working environment (Figure 3).

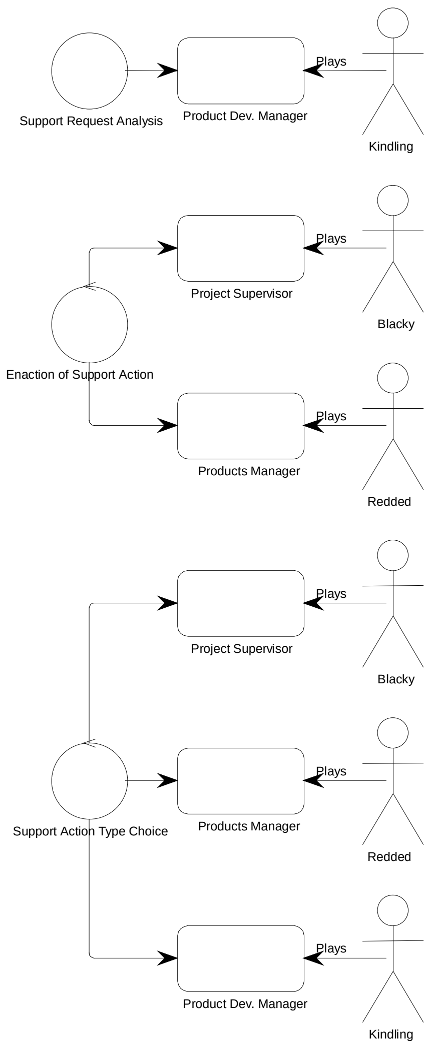


Figure 3: The communications problem

This represented a significant advancement in the knowledge of the development process itself, especially when considering the two separate units contributing to the project. In fact, the managers of the single units were aware only of the part of the process being developed in their area. When it came to comparing the two development processes, some inconsistencies were found.

The principal inconsistency was found to be incorrect communications between each unit and the customer. As it is shown in Figure 3, there are three people involved with communications with the customer. This makes it impossible to identify a single person playing the interface role with the customer.

In addition, a fourth person, who was not intended to have any direct relationship with the customer, was acting as the company's representative for minor issues which would be dealt with without even a record of them. The person would do this only occasionally, and did not allocate any time to the activity, that was thus missing in the model. This was intended to improve the project's efficiency but ultimately resulted in unnecessary repetition of unneeded tasks.

The identification of this problem was only possible by carefully analyzing the data coming from the model and the activity based costing forms. The AIDs were then used to locate the actual communication taking place between the client and each separate activity, and PIDs were used to locate the actual persons in charge of the activities.

4.2 Monitoring the process

Monitoring the software process was one of the main goals of the project, especially from the management point of view.

In fact, top manager sponsored the project as the core of a new strategic effort to understand the company not only from the financial and marketing point of view, but also from the operations management point of view. On the other hand, process monitoring through activity based costing and object oriented process modeling would give operation management a much better view over the entire development cycle, allowing the optimization of the phases and possibly better control over productivity.

However, we did not want to transform process monitoring in a means to obtain individual productivity scores, and therefore the data coming from the single persons was always aggregated on a team basis.

Table 1 shows the high-level aggregated activity profile for both operating units during two months (February and March 1998). The numbers represent the percentage of the total man power (effort) and the total infrastructure use (including non-human resources such as computers) for the two months. Note that some of the activities did not take place: this is partly due to the optimization of the communications problem (see the following section), and partly due to the fact that in the time frame considered, those activities did not take place.

Table 1: Activity profiles

Activity	**Effort (%)**	**Infrastructures (%)**
Project Management	9,43	5,90
Customer Management (1)	3,86	,67
Personalization	16,70	12,68
Infrastructures Management	,43	,00
Ad-hoc Software Development	29,38	40,29
Process Management (1)	,64	,67
Technical Development	34,71	37,46
Customer Management (2)	,00	,00
Process Management (2)	1,42	,80
Customer Management (3)	,00	,00
Conversion service	,00	,00
Residual Efforts	2,52	1,52
Idle Time	,11	,00
Transfers	,80	,00

Table 1 is useful to modelers and control managers. However, to present the data at top manager meetings, we employed a clearer representation which better conveys the information in Table 1 (see Figure 3 and Figure 4)

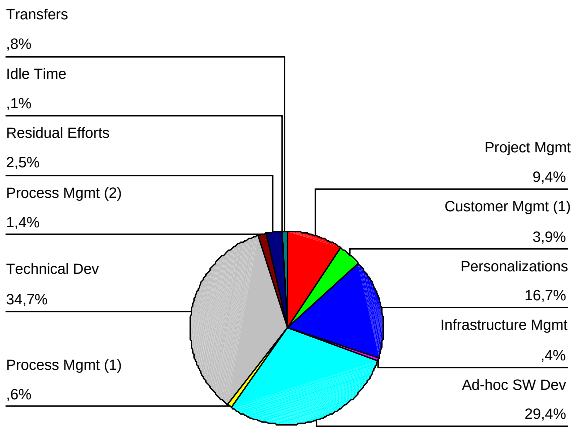


Figure 4: Effort profile chart

Moreover, this presentation was invaluable in showing the effectiveness of the modeling effort throughout the company, to establish a corporate process modeling culture, and to present valuable data to the top management.

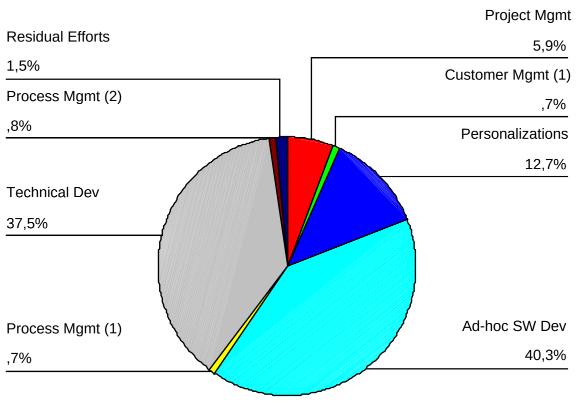


Figure 5: Infrastructures profile chart

4.3 Controlling the process

Although process control has always been the goal of the software company we analyzed, we discovered that what was currently seen as “Process Control” was an enhanced kind of project control. The process itself was not analyzed nor modified, but rather, each project manager was held responsible for keeping in synch with the main project schedule.

This way of conceiving the work has hindered true process control. For example, each project manager was accustomed to work with a small subset of the personnel in the company, and therefore would prefer resources from that small pool, regardless of any other possible equivalent choice.

To tackle this problem, the process modeling activity defined role equivalence classes based on the skills possessed by each single person in the company. This was extremely useful when, due to internal rearrangement, more than half of the people in the project we tracked had to be reassigned.

Another form of process control is shown in Table 1: there is no effort allocated for Customer Management (2) and Customer Management (3). These two activities belong to the customer relations sections in Padova and Bologna. This was the result of an explicit, informed decision that demanded the relationship with the customer to a single activity (i.e., Customer Management (1) taking place in Rome), giving it full responsibility for customer relations. This interface activity acted thereafter as the central repository for any information from and to the customer, and as a dispatcher/router for the delivery of all customer requests.

5. Conclusions

This paper presented our experience in modeling the software development process of an Italian company during the development of an information system for a large municipality. The results highlighted the advantages of understanding the process (decoding), monitoring its progress, and the new capability to control it.

Two are the main lessons learnt:

- There can not be process control without a clear understanding of the process itself. Although this seems evident, the overall business situation in Italy is that of non-standardized business processes, which are very difficult to depict, understand, and control. The modeling methodology helped identifying the software development process and highlighted the differences between the perceived process and the actual one. As a consequence of this, the teams have started coordinating with each other, and process optimizations have been possible.
- The software development process is a not a crisp entity: it fades in infinite levels of abstractions, up to the most trivial or overly general activities (such as for example “thinking”). There must be a compromise between the model’s depth and the model’s effectiveness, and this is given by the understandability by the modeling group. This does not overcome the contextual bias but provides a uniform criterion for selecting the level of abstraction required in a model. When

the criterion is adopted, it results in replicable process models: the level of detail is necessary and sufficient for replication, and the model can be shared among modelers.

There are still some open issues. The model needs to be extended to capture partial ordered sequences of events and activities. Interaction with workflow models might prove useful.

Moreover, the process measures need to be integrated with a more general view of the firm. The authors are complementing the model with a scoreboard technique that allows to obtain a “control panel” of the company.

6. Acknowledgements

This experience was made possible through the European Union European Software Systems Initiative (ESSI), through a Process Improvement Experiment grant (PIE Project No. 23699, DECO’).

References

1. Abdel-Hamid, T. The slippery path to Productivity Improvement. IEEE Software, July 1996, pp. 43-52

2. Abdel-Hamid, T. and S. Madnick. Software Project Dynamics: An Integrated Approach. Prentice-Hall, Englewood Cliffs, NJ., 1991

3. Aksit, M., F. Marcelloni. Reducing Quantization Error and Contextual Bias Problems in Object-Oriented Methods by Applying Fuzzy Logic Techniques. *Proceedings of the Modeling Software Processes and Artifacts Workshop*, ECOOP 97, 1997, Jyvaskyla, Finland

4. Bandinelli, S., A. Fuggetta, and C. Ghezzi. Software Processes as Real Time Systems: A case study using High-Level Petri nets. *Proceedings of the International Phoenix conference on Computers and Communications*. Arizona, April 1992

5. Bandinelli, S., A. Fuggetta, C. Ghezzi, and S. Grigolli. Process Enactment in SPADE. *Proceedings of the Second European Workshop on Software Process Technology*. Trodheim, Norway: Springer-Verlag, 1992

6. Bandinelli, S., M. Braga, A. Fuggetta, and L. Lavazza. The architecture of the SPADE-1 Process-Centered SEE. *3rd European Workshop on Software Process Technology*. Grenoble, France, 1994

7. Benedicenti, L., N. Morfuni, P. Predonzani, G. Succi, and T. Vernazza, The Effects of Process Modeling in a Software Engineering Firm, *Proceedings of the Acquire ICT'98 International Conference*, February 1998

8. Benedicenti, L., P. Predonzani, G. Succi, T. Vernazza. Gertrude: OO for BPR. *Proceedings of the 6th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'97)*. Los Angeles, CA, September 1997

9. Curtis, B., M.I. Kellner, and J. Over. Process Modeling. Communications of the ACM, Vol. 35 No 9, pp. 75-90, 1992.

10. Gruhn, V., W. Schafer. Software and Business Process Technology. Tutorial at ICSE 97, Boston, Ma

11. Innes, J., F. Mitchell. I costi di struttura – Metodologie di analisi e di gestione. Egea, 1994.

12. Jacobson, I., M. Ericsson, and A. Jacobson. The object advantage - business process reengineering with object technology. ACM Press, 1995.

13. Kaplan, R. S., and D. P. Norton. The Balanced Scorecard - Measures that Drive Performaces. Harvard Business Review. January - February 1992, pp. 71-79.

14. Mayer, R.J., IDEF family of methods for concurrent engineering and business reengineering applications. Technical report, Knokledge Based Systems, Inc., 1992.

15. Osterweil, L. J. Software Processes are Software Too. *Proceedings of the Ninth International Conference of Software Engineering*, pp. 2-13, Monterey, CA, 1987

16. Osterweil, L. J. Software Processes are Software Too, Revisited: An Invited Talk on the Most Influential Paper of ICSE 9. *Proceedings of the 19th International Conference of Software Engineering*, pp. 540-548, Boston, MA, 1997, ACM Press

17. Rumbaugh, J., M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. Object-Oriented Modeling and Design. Prentice-Hall, Englewood Cliffs, NJ., 1991

18. Teng, J.T.C., S.R. Jeong, V. Grover. Profiling Successful Reengineering Projects. Communications of the ACM, Vol. 41, No. 6 (June 1998) pp. 96-102, ACM press

Tailoring product focused SPI

- Application and customisation of PROFES in Tokheim -

Rini van Solingen[14], Tokheim & Eindhoven University of Technology
Arnim van Uijtregt[15], Tokheim & Eindhoven University of Technology
Rob Kusters, Eindhoven University of Technology & Open University Heerlen
Jos Trienekens, Eindhoven University of Technology, The Netherlands

Abstract

Management problems in the development of software have been addressed by a focus on improvement of the development process the last years. However, in most cases the product is sold, not the process that created it. Therefor, process improvement should have a product focus. This paper presents the practical implementation of a method for product focused software process improvement in Tokheim, and describes experiences in one development project. The main conclusions are that product focused SPI puts the product in a central position, addresses the specific needs of the company very well, and results in early availability and better visibility of benefits. As a result, commitment of the product development team to this kind of improvement programmes is established. Furthermore, it appears that the benefits of method application outweigh the (relatively low) cost already in an early phase of the project.

1. Introduction

When striving towards software product quality, two main approaches can be distinguished: the process approach and the product approach. The process approach tries to improve product quality indirectly by improving the process, while the product oriented approach tries to create product quality directly.

[14] Contact the authors via: R.v.Solingen@tm.tue.nl

[15] Currently employed by Dräger Medical Technology, Best, The Netherlands

The process approach assumes a positive correlation between software process improvement (SPI) [3] and product quality. However, when quality improvement activities focus too much on the process without being clear about the expected impact on product quality, it is possible that effort is invested in activities that barely effect product quality. Also, it is possible that the process improvement activities have effect on quality areas where the product quality is already according to user needs, while quality areas that need attention are overlooked. It is therefore important to invest in process improvement activities that focus on product quality, and to invest in those activities that have the best effect on product quality.

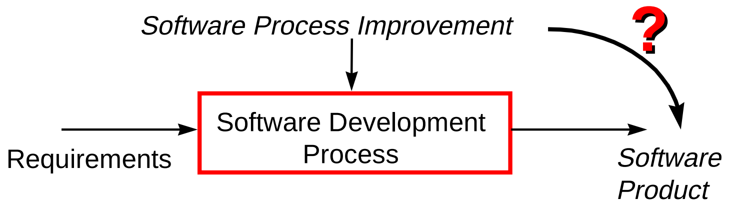


Figure 1: Relation between SPI and product quality.

This paper presents the way in which the PROFES methodology [1] is customised to Tokheim. Furthermore it contains a description of applying the approach in a development project. The PROFES methodology has been developed in the EU project PROFES (23239), which customised successful approaches into one single embedded systems specific methodology that links product quality objectives directly to the software development process. The approach presented in this paper is also based on a project of Tokheim and Eindhoven University of Technology: SPIRITS (Software Process Improvement in embedded IT environments).

1.1 Tokheim

Tokheim is world market leader in equipment and services for self-service petrol stations. Tokheim has a yearly revenue of 750 million US dollar, and 4,800 employees. Tokheim products are Fuel Dispensers, Point of Sales, EFT equipment, Back-Offices and Forecourt Controllers.

1.2 Tailored method for product focused SPI

For Tokheim the PROFES improvement methodology [1] had to be integrated in the day to day process. Therefor integration and tailoring has been carried out. To make the PROFES methodology for product focused SPI more explicit and operational, the flowchart of Figure 2 has been constructed which presents the PROFES methodology on an operational level for Tokheim.

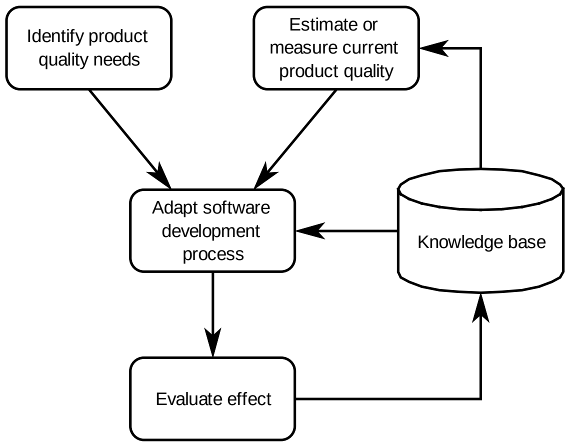

Identify product quality needs
Estimate or measure current product quality
Adapt software development process
Knowledge base
Evaluate effect

Figure 2: The tailored phases for product focused SPI.

The tailored method combines software process improvement activities with software product quality activities. The method focuses on those areas of the software development process that are not able to meet the product quality objectives. The knowledge base contains models of the expected impact of process improvement actions on product quality. Such a knowledge base is therefore essential for product focused process improvement.

The main source for process changes in this method is the comparison of product quality targets with the actual product quality or estimated product quality if there is no product. The differences between target and actual product quality are the main product quality goals that should be worked on. Process improvement actions that address these product quality goals will be selected after consultation with the project team. In this negotiation it may prove to be impossible to reach some quality targets. In such cases the target can be adjusted accordingly.

Each step in Figure 2 will be presented in detail in the next sections. Also case study results on applying these steps are provided. These results have been based on the development project that will be introduced in the following section.

1.3 Case study project description

The family of Outdoor Payment Terminal products (OPT) consists of products that provide the facility to purchase fuel without requiring the intervention of a station operator or cashier. The fuel purchaser can initiate and complete a fuel purchase transaction with the use of an OPT. For this purpose the OPT is equipped with several peripherals :

• card reader	to pay with credit, debit, or private cards.
• cash acceptor	to pay for the fuel with cash.
• user keyboard	to interact with the system.
• user display	to interact with the system.
• receipt printer	to provide a receipt of the transaction.

2. Product quality specification

The objective of this step is to set the targets for product quality. With these targets the most appropriate development process can be selected, and also product quality evaluation can be carried out once the product is created. Two approaches [6] are applied for the specification of product quality. Firstly, an embedded software specific questionnaire is used to provide a global overview on required product quality.

Secondly, a more extensive survey of product quality needs is applied. All product stakeholders that are involved with the product in some way are consulted [5]. In open interviews with these stakeholders the quality requirements are gathered. For each of the requirements metrics are specified to enable measurement. The feasibility of the requirements is discussed with the project manager. Some of the requirements are rejected which means that no effort will be spend in order to reach the requirement. The ‘identify product quality needs’-phase is presented in Figure 3.

The first step is to gain commitment. Following this, a global survey of quality needs is carried out using a questionnaire. The project manager completes this questionnaire, because the project manager is responsible for the product and is the one most likely to possess the required information. This results in a global overview of product quality needs which is used as input for the extensive survey of quality needs [5]. This survey is needed, because it generates more detailed information. Open interviews are held with the various stakeholders.

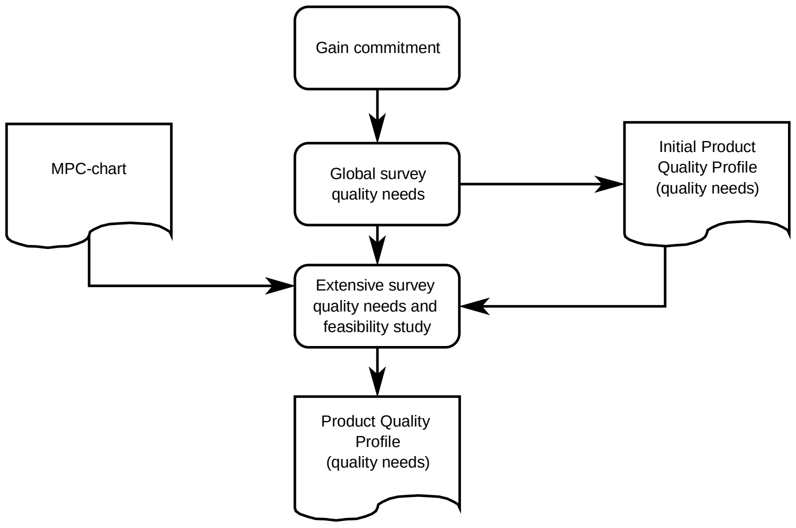

Gain commitment
MPC-chart
Global survey quality needs
Initial Product Quality Profile (quality needs)
Extensive survey quality needs and feasibility study
Product Quality Profile (quality needs)

Figure 3: Identify product quality needs phase.

The initial quality needs help to guide the interview to a certain extent, because the interviewer already has some insight in the required product quality. During the interviews quality requirements are gathered, and metrics are determined for the requirements, which enables checking whether the product meets the requirements. Later these metrics are used to evaluate the conformance of the product to the requirements. The result of this phase is a ‘product quality profile’ in which the required product quality is documented.

2.1 Case study results

During the specification of the product quality targets, 62 requirements were stated during four interviews with product stakeholders. These requirements were specified in natural language, classified over the ISO 9126 quality characteristics [4], prioritised, evaluated by the project manager, and specified with metrics. For each metric a current value, ideal value, and target value was specified as well.

3. Product quality assessment

The objective of this step is to make the current product quality explicit, in order to compare it with the product quality targets. The differences between current and wanted product quality identify improvement areas for the product. If there is no version of the product available, estimation is carried out on what the product quality will be using the ‘normal’ process.

Product quality assessment can be done in two ways:

1. Measure current product quality. In other words, evaluate the extend to which the product complies with the product quality targets.

2. Estimate product quality. Make a prediction what product quality will be when using the ‘normal’ process. This presupposes an understanding of the current process. A development process model can be used to support this task, as can the result of a process assessment.

The PROFES method uses SPICE conformant assessments (e.g. BOOTSTRAP [2]). Part of the assessment is the investigation and identification of the current way of working, i.e. how the software development process functions at this moment. The assessment report contains processes and activities of the development process. By applying the process-product relationship models from the knowledge base (Figure 2) the contribution to product quality for each specific process or activity can be estimated. Therefor it is possible to estimate the product quality that is created by following the ‘normal’ process.

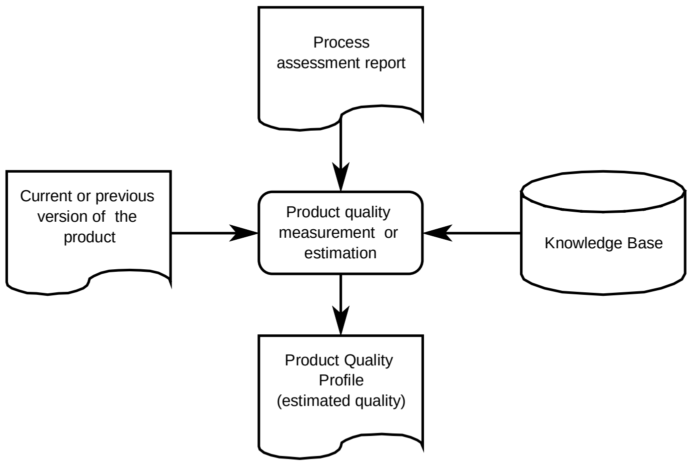

Process assessment report
Current or previous version of the product
Product quality measurement or estimation
Knowledge Base
Product Quality Profile (estimated quality)

Figure 4: Estimate or measure current product quality phase.

3.1 Case study results

The process assessment of the OPT project identified that 34 actions are currently taken in the software development process which impact product quality. An estimate was made for product quality by applying the process-product relationship models from the knowledge base, together with a description of these 34 actions. This estimation represented the product quality that would probably be reached when using the current process, as long as nothing was changed.

In Figure 5 both the product quality targets and estimated product quality are presented. This figure presents the quality profile of the OPT product along the six product quality characteristics [4]: functionality, reliability, usability, efficiency, maintainability and operability.

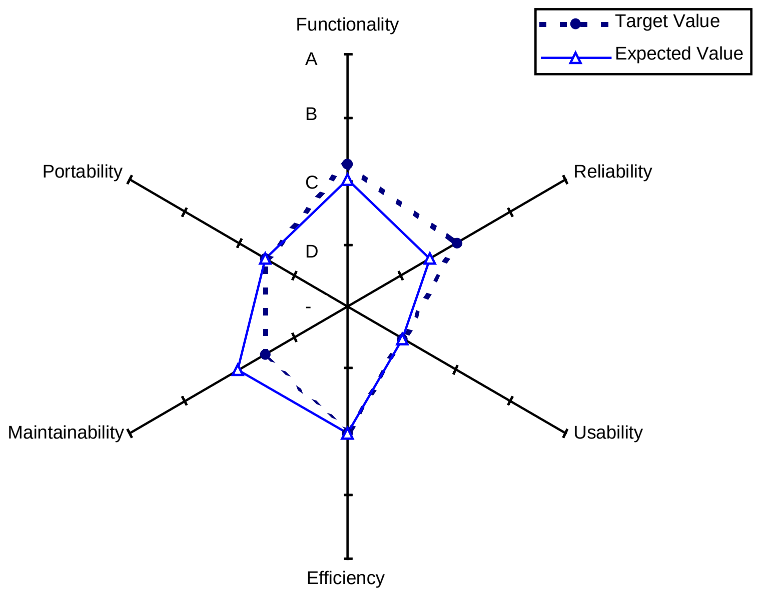
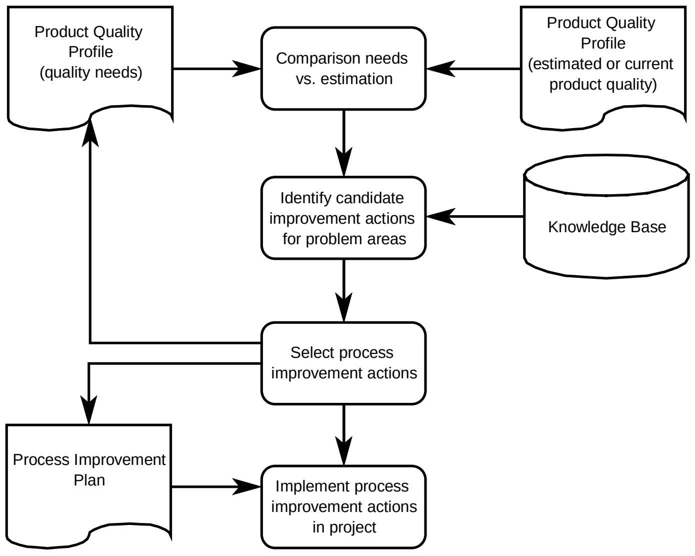


Figure 5: Quality needs vs. estimated quality for the OPT.

4. Changing the development process

Based on the differences between current product quality and the product quality targets, the development process can be changed. Decisions will be taken on where to change the software development process, in order to arrive at a process that is capable of creating the required product quality. Based on the product quality targets, an analysis is done to tailor the software development process to the product quality requirements.

The ‘adapt software development process’-phase is presented in Figure 6. It starts with an analysis of the product quality targets from the previous phases, in order to identify problem areas for which the development process is not sufficiently suited to meet the quality needs. For these problem areas, candidate improvement actions are identified. In Tokheim a tool is available that contains process-product relationship models. Finally, the Process Improvement Plan is created and carried out.



Product Quality Profile (quality needs)
Comparison needs vs. estimation
Product Quality Profile (estimated or current product quality)
Identify candidate improvement actions for problem areas
Knowledge Base
Select process improvement actions
Process Improvement Plan
Implement process improvement actions in project

Figure 6: Adapt software development process phase.

4.1 Case study results

Figure 5 presented the quality profile of the OPT product along the six product quality characteristics [4]. The main conclusion from Figure 5 was that for the product quality characteristics: usability, efficiency and portability the current process was likely to reach the quality target. For maintainability the current process would probably do better. However, two problem areas were identified:

1. It was expected that functionality (in this case the sub-characteristic suitability) of the product was at risk. This was mainly because there were many unknown country specific requirements, to which the product had to comply.

2. It was expected that reliability (especially the sub-characteristic maturity) might not be sufficient. In the development process actions were taken that should improve reliability, but these actions were not executed completely.

Based on these findings, a meeting with the project team was held in which the results of the investigation were presented. Also candidate improvement actions to address the improvement areas were proposed in this meeting. The following decisions were taken during that meeting.

- Action is taken by marketing to collect the country specific requirements.
- The project team members were surprised by the analysis with respect to reliability , because in their perception the customers of the development team are quite happy with the reliability of the product. Measurement of product reliability in the field is therefor started.
- Also for the OPT reliability, there is a large dependence on service department product testing. In order to identify the extend in which this is actually done, a system test checklist for the service department will be created.

5. Evaluation of effects

After the software development process definition and the start of the project, the effect on process and product quality is evaluated. The evaluation is done in two ways. First, the evaluation of product quality is carried out. This evaluation identifies whether the current version of the product meets the quality needs. Second, the evaluation of impact of certain process improvement actions on product quality is carried out.

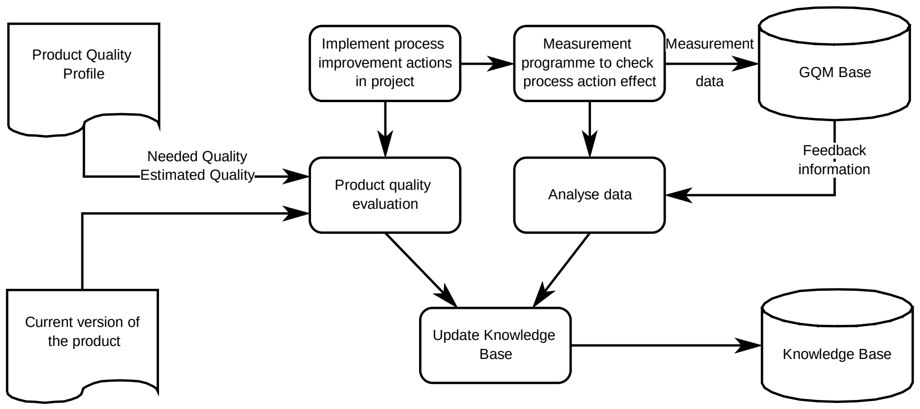

Product Quality Profile
Implement process improvement actions in project
Measurement programme to check process action effect
Measurement data
GQM Base
Needed Quality
Estimated Quality
Product quality evaluation
Analyse data
Feedback information
Current version of the product
Update Knowledge Base
Knowledge Base

Figure 6: Evaluate effect phase.

Goal-oriented measurement programmes [7][8] are designed to measure process and process aspects for the above listed evaluation purposes. The detailed steps of this phase are depicted in Figure 6.

5.1 Case study results

The final phase is the evaluation of the effects of the process improvement actions and the evaluation of the product quality at the end of the project. Although the OPT project is not finished yet, one example is presented from the set of available measurement material.

Product reliability was measured by looking at the distribution of failures after release. Figure 7 shows that in total 43 problems have been reported. The distribution is shown over failures reported per month.

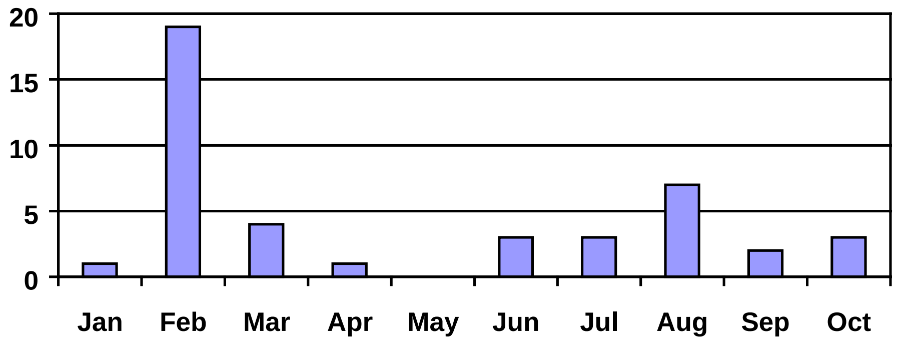


Figure 7: Distribution of failure reports over months.

The main conclusion drawn by the project team from these first measurements was that OPT reliability was sufficient. The measurements would be continued, especially when the OPT was installed in the field. Other measurements were taken also to identify other product reliability details.

6. Cost/Benefit analysis of the case study

A cost/benefit analysis has been carried out on method application in the OPT project. Cost can be deducted by measuring the effort in person hours, but it is very difficult to express benefits in financial terms, because they are often indirect and not objectively measurable. For example, a sales increase is rarely caused by increased product quality alone.

The cost/benefit analysis will be done as follows. The calculated cost (measured in man hours) will be valued against both financial and non-financial benefits.

6.1 Cost

The costs are measured in person hours. Table 1 contains the effort spent on method application in the OPT project.

Table 1: Effort needed for the OPT case study (in person hours).

Phase	Quality Engineer	Quality Manager	Project Manager	Software Engineers	Total
Identify product quality needs	50	4	8	10	72
Estimate product quality	4				4
Adapt software development process	30	1.5	1.5	3	36
Evaluate effect	79	1.5	4	10.5	95
Total	163	7	13.5	23.5	207

In total 207 hours have been spent on the quality activities, of which only 18% were spent by the project team. During the OPT project the project team spent about 1% of their time on these quality related tasks.

6.2 Benefits

It is difficult to relate specific benefits to method application directly and express them in financial terms, such as increased sales or reduced cost. This is especially difficult, because the application of the method has not been completed yet and the project is still running.

However, benefits can be linked to the influence of the application of the method and can be determined in non-financial terms. These benefits can then be balanced against the costs. The benefits are divided in direct and indirect benefits. The direct benefits are benefits that are directly related to the method. The indirect benefits are the positive side effects of method application.

The direct benefits of the OPT case study include:

- The quality requirements for the OPT product are made explicit and measurable.
- Strengths and weaknesses of the development processes were discovered.
- Based on the differences between the quality needs and the strengths and weaknesses of the software development process, problem areas have been identified for the project, where the development process is expected to be insufficient to meet the quality needs.
- The main national requirements for the OPT are now available, after effort was being invested as a result of the quality investigation.
- For the OPT a test checklist is created for the service department, which is expected to improve the testing process and in that way OPT reliability.
- The status of OPT reliability is now much clearer due to the measurements.

The indirect benefits of the OPT case study includes:

- The co-operation between quality assurance and the OPT project team has improved considerably.
- Direct attention and priority to software quality has increased quality awareness in the project team.

6.3 Are the benefits worth the cost?

In the previous two subsections the cost and benefits of both projects are presented. The question remains whether the benefits are worth the cost.

Because of the improved relationship between the quality assurance department and the OPT development team, the benefits seem to outweigh the cost, considerably. The cost, 207 hours, are low. The increase of attention towards quality by the project team, and the co-operation with the quality assurance department are major benefits. In this light, the effort has to be split into: effort of the quality assurance department and effort of the project team.

The cost for the project team consists only of 37 person hours, which is very low. The effort of the quality assurance department, 170 person hours, would otherwise be spent on quality activities like ISO 9001 audits and process assessments. The advantage of the new approach is that the quality assurance department co-operates much more with the project team, which is experienced as very beneficial. Also the gathering of quality requirements, the creation of the checklist and the insights in OPT reliability contribute to the justification.

7. Conclusions

This paper has presented the results of tailoring the PROFES methodology to Tokheim, illustrated with experiences of a real-life development project.

The main conclusion is that product focused SPI puts the product in a more central position. Therefor the specific product needs of the company are well addressed, and benefits are available soon and well visible. As a result, high commitment of the product development team to this kind of improvement programmes was established. The PROFES methodology was experienced as highly applicable, and tailoring it to Tokheim was carried out easily. Furthermore, it appeared that the benefits of applying the method outweighed the (relatively low) cost already early in the case study. It is expected that the benefits will still increase over time.

8. Acknowledgements

The authors would like to thank the engineers and managers responsible for the OPT development project for their participation and enthusiasm. Also, thanks to all partners in the PROFES and SPIRITS projects. Furthermore, the authors thank Erik Rodenbach, Shyam Soerjoesing, and Roy de Jonge for their contributions to the results presented in this paper.

References

[1] Birk, A., Järvinen, J., Komi-Sirviö, S., Oivo, M., Pfahl, D., PROFES – A Product-driven Process Improvement Methodology, Proceedings of the Fourth European Software Process Improvement Conference (SPI '98), Monte Carlo, 1998.

[2] Bicego, A., Khurana, M., and Kuvaja, P., "BOOTSTRAP 3.0 – Software Process Assessment Methodology". Proceedings of the SQM '98, 1998.

[3] Humphrey, W.S., Managing the Software Process, Addison Wesley Publishing Company, 1989.

[4] ISO/IEC, ISO 9126: Information Technology - Software quality characteristics and metrics, ISO/IEC, 1996.

[5] Kusters, R.J., Solingen, R. van, Trienekens, J.J.M., User-perceptions of embedded software quality, Proceedings of the Eighth international workshop on software technology and engineering practice (STEP'97), pp. 184-197, London, July 14-18, 1997

[6] Kusters, R.J., Solingen, R. van, Trienekens, J.J.M., Strategies for the identification and specification of embedded software quality, to be published in: Proceedings of the Tenth international workshop on software technology and engineering practice (STEP'99), 1999.

[7] Latum, F. van; Solingen, R. van; Oivo, M.; Hoisl, B.; Rombach, D.; Ruhe, G., Adopting GQM-Based Measurement in an Industrial Environment, IEEE Software, January 1998.

[8] Solingen, R. van, Berghout, E., "The Goal/Question/Metric method: a practical guide for quality improvement of software development". McGraw-Hill, 'http://www.mcgraw-hill.co.uk/vansolingen', ISBN 007 709553 7, 1999

SESSION 7:

Process Modelling and Assessment

Software Process Improvement in Small Organizations Using Gradual Evaluation Schema

Naji Habra
Eustache Niyitugabira,
Anne-Catherine Lamblin
and
Alain Renault
Institut d'Informatique
Technology Transfer Center
University of Namur
Namur, Belgium

Abstract

This paper relates a technology transfer experience which aims at supporting the introduction of software process improvement in small businesses, small organizations and/or small projects. The experience is born from a European interregional collaboration between two university research teams (France and Belgium) and a public technology center (Luxembourg). One of the contributions of this experience is the design of a Software Process Improvement approach particularly adapted to small units on the one hand, and to regional context, on the other hand. The proposed approach is gradual. It is based on three nested evaluation models ranging from an extremely simplified model (the micro-evaluation model) to a complete standard model supporting SPICE. The intermediate model, called the mini-evaluation model, can be viewed as a tailoring of SPICE and can be used by itself as a definitive model by small businesses and small organizations.

1. Context and Motivation

The project is mainly addressed to the Small and Medium Enterprises (SMEs) and small public organizations of the Walloon region, i.e., the French speaking part of Belgium, which is one of the oldest industrial region in Europe. Similarly to other old European industrial basins, the region suffers from heavy aged industrial structures, e.g., iron and steel industry, coal-mining… The region is achieving a phase of slow conversion to modern industrial structures including small businesses which are active, among other, in the domain of Information Technology (IT).

The main characteristics of the regional environment are the persistence of some old-fashioned bureaucratic management style, the coexistence of new small dynamic businesses and old big industries, the small size of IT businesses and the very small size of the majority of IT units in other industries and in public organizations. A regional study made by the Technology Assessment Group (CITA) of our university about Walloon SMEs [1] gives some significant data: in about 30% of businesses, only one person has software (in general) in his charges; and among the SMEs developing and/or managing Information Technology, 60% achieve these tasks with less than 5 persons. Such a very small size makes businesses highly dependent on some projects, some actors and/or on some technical capabilities, though they could be sometimes very innovative in their domains.

Another characteristic of the SMEs of that region lies in the fact that they are surrounded by rapid growing dynamic regions (French Lorraine Region, Grand Duchy of Luxembourg,…) and they evolve in a European context where the market is more and more open, and consequently, with an increasing competition. In this context, it is obvious that software quality in general becomes a crucial issue for Walloon SMEs even though their resources are very limited.

The OWPL[16] project, supported by a public funding of the Walloon region, aims at assisting SMEs in their Software Process Improvement (SPI). In particular, the main goal is to provide SMEs and small public organizations with very simplified adequate models to initiate SPI approaches.

In fact, standard models like CMM were initially designed for bigger structures. So, they should be, more or less deeply, tailored and/or adapted to very small organizations like our target SMEs. The first reason is the cost of an evaluation process (+/- 25000$) and its duration (+/- 8 month) [2] which are disproportional to the available resources. In addition, the maturity level our target SMEs would get according a general assessment model like CMM, would be very low. Brodman and Johnson ([3],[4]) show that a great number of process improvement plans based on the CMM encountered problems and that an important rate of those problems (53%) were related to the size. The success of a CMM process improvement plan actually grows with the number of people having software process in charge.

There is also a similar need of adaptation with the SPICE model, even though this model is intended to be suitable to SMEs. The cost and effort remain too much important for very small organizations. A very simple adapted model would be more suited for them (at least) as a starting point.

Another important point, lies in that the number of actors involved in software process is very small. Several roles can be in charge of the same single person . This makes the use of such models very complex for small organizations.

In addition, actors in SMEs are far from being all Software Engineering specialists ; so adapting the vocabulary is necessary to allow the model to be used for self- assessment or for an assessment with a light support.

[16] The acronym OWPL stands for *Obsrevatoire Wallon des Pratiques Logicielles*, i.e., Walloon Observatory for Software Practices .

In summary, regional SMEs have a critical lack of software process improvement in order to be competitive in a short or medium term. But, due to their very small sizes and their limited resources, they need an adapted model they can put in practice immediately and in a simple way.

The remainder of this paper describes the experience of the OWPL project whose aim is namely to produce and experiment such a tailored model. The project is undertaken by the Technology Transfer Center of the university of Namur and funded by the Walloon Region (Belgium). Meanwhile, our center collaborates with the University of Nancy (France) and the Center of Public Research of the Grand-Duchy of Luxembourg in a European ESSI project SPIRAL*NET[17]. This project has the more general goal to increase the visibility of regional SMEs and to improve the SMEs software process in general by the generalization of their best practices. The target of the European project is the French speaking area composed of the Grand Duchy of Luxembourg, the Walloon part of Belgium and the French Lorraine.

2. The OWPL Approach

The main original idea of the OWPL approach of software process evaluation and improvement is to proceed using three nested models which can be used either separately or as successive stages in the SPI.

1. A first extremely simplified model (called the micro-evaluation model) which is designed to have as lower cost as possible but also to allow giving a first pertinent diagnostic to the assessed organization. The rationale is twofold, to make the assessed SME aware of its weakness but also of the potential effective improvement it can expect, on the one hand, and to determine the priorities of subsequent stages of evaluation and improvement procedures, on the other hand.

[17] SPIRAL*NET is the ESSI ESBNET project 27884.

2. An intermediate model (called the mini-evaluation model) which is the core of the OWPL approach. This model can be viewed as a tailoring of SPICE model (with significant influence of CMM and Bootstrap) particularly adapted to the context described in the above section. This model can be used by itself and would be sufficient for the majority of small businesses and small organizations. It can also be used as a stage that prepares a full evaluation according to one of the standard models.

3. The third model is the evaluation model we propose to organizations having a certain maturity level and seeking for a more in depth evaluation of one or more selected processes in reference to an international standard . In such cases we propose the use of the SPICE model.

Hereafter we give some details about the three nested models we propose.

2.1 The micro-evaluation model

The aim of the micro-evaluation is to give a first outlook of the evaluated organization, to make a diagnostic and guide the next steps of software process improvement. The main requirement that drives the design of this model is to be as less costly as possible, in time and money.

So, the designed model corresponds to a half an hour interview based on a well-prepared questionnaire. The questionnaire covers six key axes we select as the most pertinent and the most prior to our target organizations on basis of former experience with SMEs evaluation.

These axes are the following:

1. quality assurance,
2. customers management,
3. subcontractors management,
4. project management,

5. product management, and
6. training and human resources management.

The questionnaire includes a few dozens of questions covering the axes above. Questions are open, and each of them is associated with one or more sub-questions allowing the interviewer, if need be, to adjust and refine the information he gets. Evaluations are performed by members of our software quality team, the interviewed person should be the one who has the software quality in his charges in the evaluated organization ; this corresponds usually to one of the executive staff members or to the quality engineer, if this function exists.

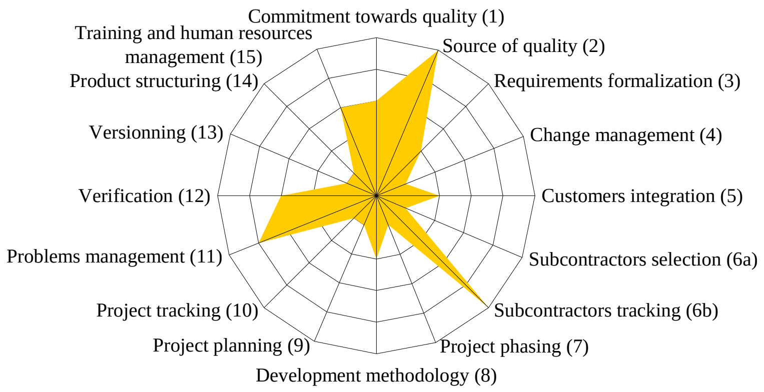
Answers are interpreted according to a fixed grid. Two types of questions can be distinguished. On the one hand, questions that concern essential practices related to the general organization are rated on a linear scale according to the quality of the practice assessed. On the other hand, questions that concern the software practices are rated in a double-entry grid according to the quality of the practice and to its effective implementation in the evaluated organization (only for some critical projects, for all projects,...). Detailed description of the micro-model can be found in [13].

The result of the micro-evaluation is drawn up in a report of a dozen of pages. A typical report first presents briefly the approach, then it develops the results of the questionnaire and summarizes them according to the six axes, then it analyses those results according the situation of the evaluated organization (the age, the history, the declared goals,..) and finally gives some recommendations to help the assessed unit to improve.

The micro-model has been experimented on a sample of two dozens of representative organizations (IT small companies, IT services in other businesses, public administrations using IT). Figures 1, 2 and 3 below give examples of the resulted grids for three different situations. The first grid is the detailed evaluation results according to the selected practices while the second one is a summarized pictures according to the six selected axes.

One can notice, that the first two cases show an evident weakness in the process of software development itself. This corresponds actually to an amateurish development without any well-distinguished phases or even any notion of a lifecycle. Though, these two units have some strengths in the subcontractor management, for example. A software process improvement for these units should obviously start by the elaboration of a lifecycle and of a development methodology.

The third example corresponds to a more mature unit which can expect, in the short or the middle term, a good evaluation according to a more complete model. Some weaknesses in the given assessment correspond, in fact, to some good practices which are applied only to some projects but not generalized to all the projects.



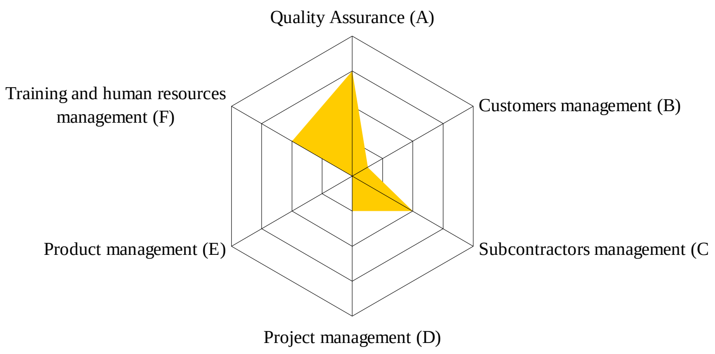
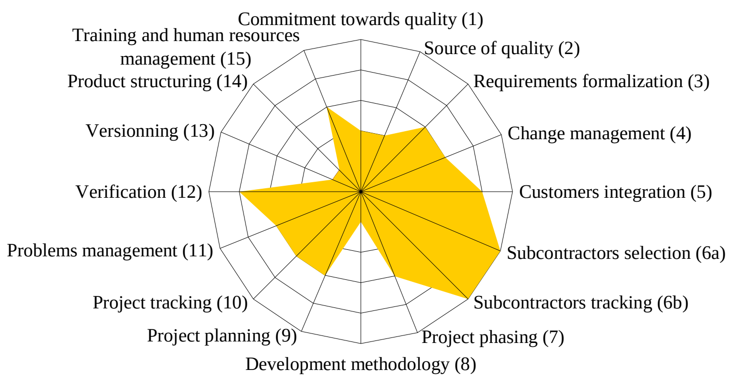


Figure – 1


Commitment towards quality (1)
Source of quality (2)
Requirements formalization (3)
Change management (4)
Customers integration (5)
Subcontractors selection (6a)
Subcontractors tracking (6b)
Project phasing (7)
Development methodology (8)
Project planning (9)
Project tracking (10)
Problems management (11)
Verification (12)
Versionning (13)
Product structuring (14)
Training and human resources management (15)

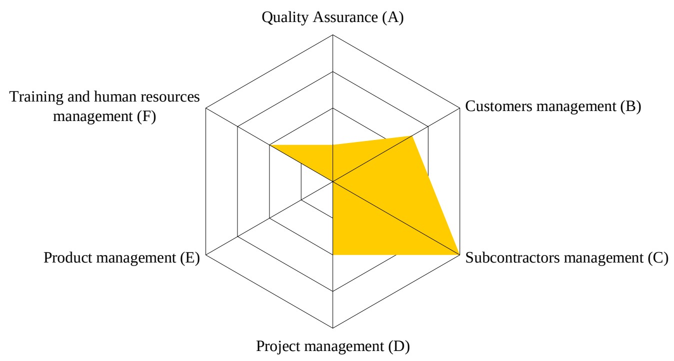
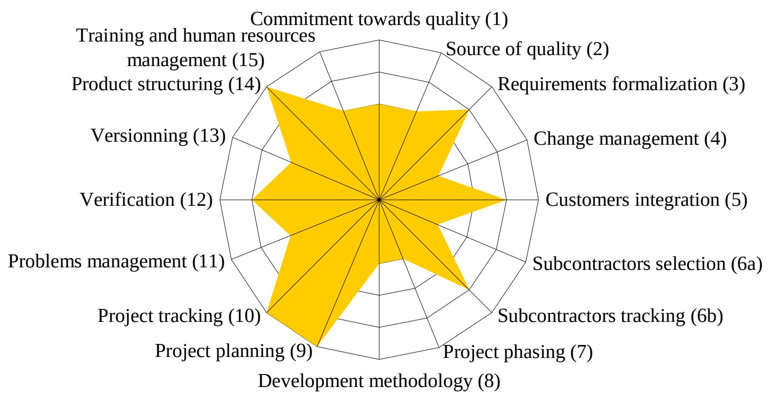

Quality Assurance (A)
Customers management (B)
Subcontractors management (C)
Project management (D)
Product management (E)
Training and human resources management (F)

Figure – 2



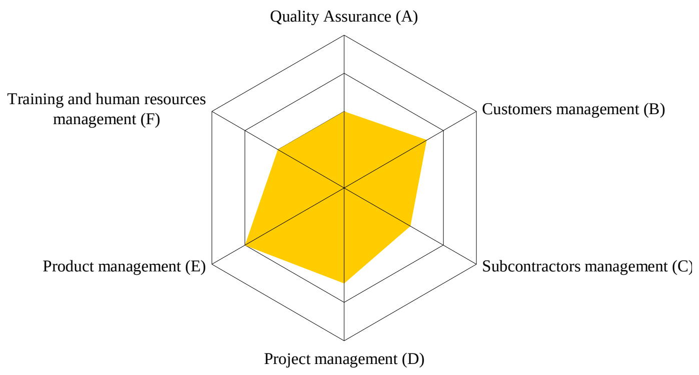


Figure – 3

2.2 The mini-evaluation model OWPL

The mini-evaluation model is the main task of the project OWPL which aims at adapting quality models, e.g., CMM and SPICE, to the context of the regional SMEs described in Section 1. The micro-evaluation model above could be viewed as a preparatory phase preceding the use of the mini-evaluation model. This latter one should be sufficient by itself for the majority of small organizations.

2.2.1 The adaptation principles

The adaptation of standard models that underlies the elaboration of the OWPL tailored model follows the key ideas below.

– The OWPL model is mainly based on a tailoring of SPICE but it is also influenced by CMM and Bootstrap. A certain traceability between the tailored model and SPICE is preserved.

– The tailored model focuses on *evolution* aspects rather than *evaluation* ones. In fact, our target SMEs would probably get a very low CMM maturity level, for example. Though, they need to know their strengths and weakness and they particularly need guidelines to improve their process.

– The tailored model uses a simplified vocabulary and avoid as much as possible the use of technical terminology. In fact, certain terms used in the classical models (or at least in their translation to French) appear too technical and troublesome.

– More generally, forms and questionnaires are simplified to avoid the model utilization to appear as a cumbersome task which involves extra bureaucratic work (see e.g., [5]). Small business resources are too few and necessary to their immediate production tasks.

– Different tasks in the different practices of the model are considered to be possibly (and even likely) assigned to the same person. So, the model clearly indicates that the different terms used for the tasks description designate different roles but not necessarily different persons. The fact that two different tasks are (or must be) assigned to different persons should be given explicitly.

– The model emphasizes the importance for an organization to define explicitly its objectives in general and those of its software process in particular. The model invites the assessed organization to refine its objectives into goals and sub-goals and to relate them to the processes and the practices of the OWPL model. We believe that making explicit the relationship between the outcomes of processes and practices on the one hand, and the declared goals of the organization on the other hand, would be motivating in the improvement process. The importance of making explicit the definition of goal is pointed out by the GQM approach [6][7].

– The model is associated with methodological guidelines concerning the action of software process evaluation as well as the awareness actions, the communication of results,..

2.2.2 The structure of the mini-evaluation model OWPL

Practically, the structure of OWPL model involves processes, practices and success factors (see Figure-4 below).

The mini-evaluation model OWPL defines 8 processes each decomposed into a number of practices (between 3 and 12) and is supported by some success factors. The OWPL processes are issued from the SPICE and CMM ones by assembling and simplification. In particular, a number of high-level practices are regrouped in a process called “capitalization and leveraging”. This process actually includes all practices related to the generalization of acquired basic practices and their utilization in order to improve in the medium term and the long term.

The identified processes are thus the following ones:

1. quality assurance process,
2. requirements management process,
3. configuration management process,
4. subcontractors management process,
5. development process,
6. project planning process,
7. project tracking and oversight process,
8. capitalization and leveraging process.

Each of the above processes is assigned a general goal in accordance with the organization defined objectives. It involves a number of practices and is supported by a number of success factors. One can notice the traceability between the above process and the key axes used in the micro evaluation model (Section 2.1).

Each practice is defined by its goal, its inputs and outputs, the resources assigned to support it and its weight. This last attribute is an indicator of the importance of the practice for the whole process improvement, its possible values are *normal*, *high* or *critical*.

Success factors are general requirements related to the environment of the process which determine its effective success. They correspond in fact to CMM *Common Features*, or to SPICE *Attributes*. They includes organizational, management, technical and human resources factors. A detailed description of the OWPL model can be found in [8].

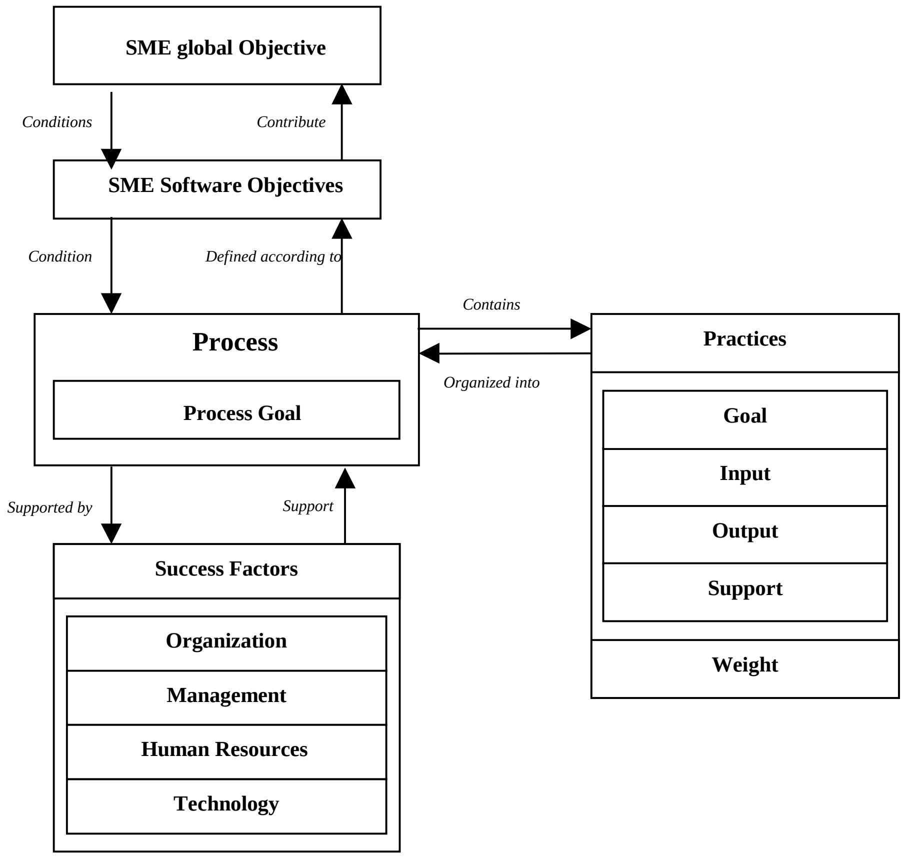


Figure 4 : OWPL Model Architecture

2.3 The complete evaluation model

Some evaluated organizations may have (or may reach) a sufficient maturity level that allow them to expect a good rating in the scale of recognized models; such rating could also be critical for them to maintain or to advance their situation in a highly competitive market. We do not actually develop a new complete model for such situations, instead we propose a SPICE evaluation focused on *some* processes which have been identified (by means of the mini-evaluation) to be the most critical ones. SPICE , as an international standard, is attractive for those SMEs seeking for recognition. Actually, at this stage of our experience, a very small number of SMEs are already at such an appropriate level.

OWPL in Practice

This section summarizes our experience with the OWPL approach. In practice, the project duration is three years, the final goal is to provide the sought tailored model and to propose it as a candidate to become a standard with some regional recognition.

The strategy we adopted is cyclic. The idea is to produce a first release of the models (for micro and mini-evaluations), then to experiment them on some representative case studies, to refine them, to re-experiment them again, and so on.

Practically, we started with an awareness action where the regional SMEs were invited to a conference-debate on Software Quality. The important audience at this conference confirmed us in our believe about the situation of regional SMEs and their demand of software process improvement. The next step was the experimentation of the micro-evaluation model on the demanding organizations. The sample is composed of above 20 organizations and includes administrative units, IS services in medium size businesses, small companies providing computer related services and/or electronics components. The experience showed that the micro-evaluation model is very attractive as a tool to start with, mainly because of its extreme simplicity. All of the assessed organizations declared to be happy with the results and the great majority demanded to continue the SPI with our team, either through a second micro-evaluation, through personal guidance, through the supply of information on SPI subjects or through a mini-evaluation. We are now experimenting the OWPL mini-evaluation model on a number of organizations which have been evaluated according the micro-model. A new release of the model taking into account the current experimentation is planned for the next September.

Bibliography

[1] (CITA) Cellule Interfacultaire de Technology Assessment, “Utilisation des Systèmes d'Information Inter-Organisationnels [SIO] par les PME Belges”. *Research Report* of SIO, Cita-Computer Sciences Dept., University of Namur, Novembre 1997. *In french*

[2] Basque R. «CBA-IPI : How to Build Software Process Improvement Success in the Evaluation Phase ?». In Software Process Newsletter, IEEE Computer Society, No. 5, pages 4-6, Winter 1996.

[3] Brodman J. G. & Johnson D. L., “What Small Businesses and Small Organisations say about CMM ?" *Procedings of the 16th International Conference on Software Engineering*, Sorrento, Italy, May 1994.

[4] Johnson D.L. & Brodman J.G., “Tailoring the CMM for Small Businesses, Small Organizations, and Small Projets” *Software Process Newsletter*, N° 8, Winter 1997, IEEE

[5] Bach J., “The Immaturity of the CMM” in *American Programmer*, Sept. 1994.

[6] Basili V.R., Caldiera G. and Rombach H.D., “Goal/Question/Metric Paradigm.” In Marciniak J.J., editor, *Encyclopedia of Software Engineering*, volume 1, pages 528-532. John Wiley & Sons, 1994.

[7] Rombach H.D., “Practical benefits of goal-oriented measurement”. In Fenton N. and Littlewood B., editors, *Software Reliability and Metrics*. Elsevier Applied Science, London, 1991.

[8] Habra N., Niyitugabira E. and Renault A. “Modèle OWPL: Evaluation et Amélioration des Pratique Logicielles dans les PME Wallonnes” Rapport Technique 1/99, Institut d’Informatique, University of Namur 1999, *in french.*

[9] Humphrey W.S., “Managing the Software Process”, SEI Series in Software Engineering, Addison-Wesley, 1991.

[10] Koch G., “Process Assessment: the Bootstrap Approach” *Information and Software Technology*, Vol30, N°6/7, 1993.

[11] Mark C. Paulk, Bill Curtis, Mary Beth Chrissis and Charles Weber Capability Maturity Model for Software, Version 1., SEI, CMU/SEI-93-TR-24, Pittsburgh, Pa, Feb. 1993.

[12] ISO/IEC JTC 1/SC 7, ISO/IEC TR 15504, 1998

[13] Renault A. “Micro-évaluation des Pratique Logicielles ” Etude de cas, Institut d’Informatique, University of Namur 1999, *in french*

Process Re-engineering Patterns

Masao Ito
Nil Software Corp.
2-17-7, Kinuta, Setagaya-ku, 157, JAPAN

Kouichi Kishida
Software Research Associates, Inc.
3-12, Yotsuya, Shinjuku-ku, Tokyo, 160, JAPAN

Abstract

In this paper, we describe process re-engineering patterns (PRPs). Process re-engineering includes re-configuration of process structure. The purpose of PRP is to provide knowledge needed for such operation as a set of patterns. In the first chapter of the paper, various concepts related to software process are clarified. And then, four fundamental PRPs are described in the second chapter.

1. Introduction

1.1 Process Orientation

The basic strategy for software process improvement is as follows:

(1) Explicitly describe the process (Process modeling as a standard).

(2) Observe the process status (Quantitative process monitoring).

The first step corresponds to the level 2 or 3 in CMM, and the second step corresponds to the level 4. During the past decades, a number of process modeling and/or monitoring methods have been developed and studied (e.g. [1]).

On the other hand, there has been almost no effort to study process re-configuration and analyze its impact. Process re-configuration is a technique needed for rather non-continuous innovational process optimization than continuous successive process improvement.

Of course, there are many best practice reports. But, most of these reports are just success stories of the introduction of some tool/methods written from the viewpoint of managers. There is no discussion from pure process-oriented viewpoint.

1.2 Definition of Terms

At first, we define some terms used in this paper. PROCESS is "all of real world activities to develop/maintain software products". ACTIVITY is "a step in the process, which creates externally recognizable status change". In this paper, process is treated as a more abstract concept than activity. Activity is related to the roles of agents (human participants or tools) or products.

In some cases, activity is also treated as an abstract concept. Let's think about a development activity performed by multiple persons. For example, "coding activity" is broken down into "developer d_1 codes module m_1" and "developer d_2 codes module m_2", etc.

Like the distinction between class and instance in the object oriented methodology, there is the distinction between a generic "coding activity" in an abstract sense and a specific "coding module_m activity by developer_d" as an instance.

In the instance-level, there are two types of relationships between activities: the relationship with other instances of the same class activity and the relationship with instances of other class activities. These two types of relationships can be represented two-dimensional network diagrams[18].

1.3 Process Economics

In this paper, process economics means the discussion about profit/loss of process planning/execution/management only based upon process structure. As defined in the previous section, instance activities have a structure shown in following diagram.

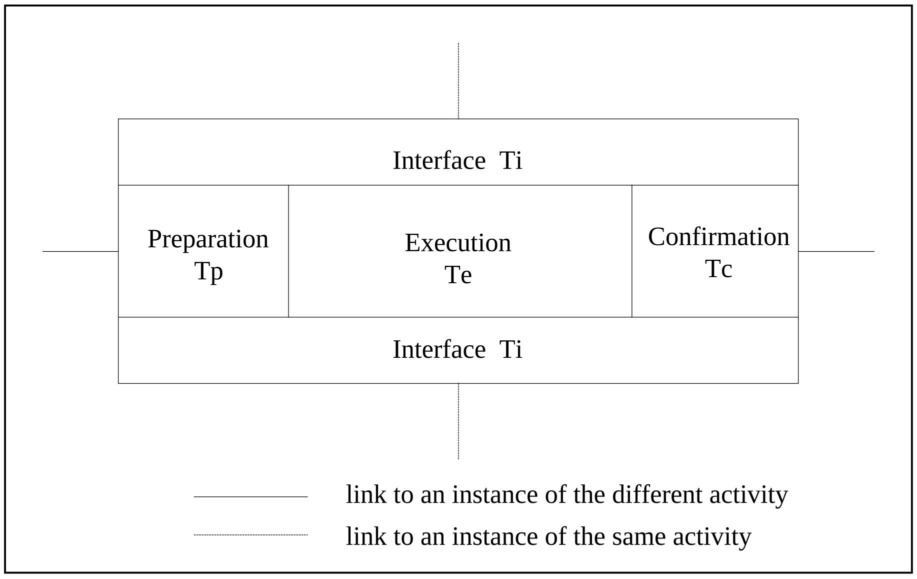

Interface Ti
Preparation Tp
Execution Te
Confirmation Tc
Interface Ti
link to an instance of the different activity
link to an instance of the same activity

Figure 1. Structure of Instance Activities

As shown in this diagram, it is possible to distinct the body of an activity to achieve its goal from other necessary operations like the preparation (interaction with foregoing activities, planning for execution, etc.) and the confirmation

[18] In the class-level, the relationship between activities becomes a two dimensional network. But, this network only describes constraints of activity execution. In the instance level, the third dimension of activity decomposition is added to this picture.

(verification of products, preparation for delivery, etc.). Also, it is possible to recognize the interfaces between other activities of the same class.

In general, it can be said as follows:

- The number of the instances is related to the possibility of decomposition of the activity.
- The number of instances is proportional to the possibility of monitoring (through confirmation).
- The longer execution time of the instance, the more risk of the schedule delay.

Therefore, from the viewpoint of process execution, it is desirable to increase the number of instances. But it is necessary to consider about the overhead of Tp, Tc, Ti,

Let's think about two class activities A and B. Each has instance activities a_1, a_2, and b_1, b_2. If we merge A and B into a single activity C, the execution time will be same:

$$Te(c_1 + c_2) = Te(a_1 + a_2) + Te(b_1 + b_2).$$

But the preparation time Tp and the validation time Tc are expected to decrease, because

$$Tp(c_1 + c_2) < Tp(a_1 + a_2) + Tp(b_1 + b_2), \text{ and}$$

$$Tc(c_1 + c_2) < Tc(a_1 + a_2) + Tc(b_1 + b_2).$$

When an activity is finished, some kind of product is delivered and should be validated. So, activity merging decreases the opportunity for monitoring.

Next, let's think about merging instance level activities. For example, merging a_1 and a_2 into single instance a_{12}. In this case, Ti (a_{12}) becomes zero and Ti (a_1) + Ti (a_2) will be included within Te (a_{12}).

It is expected that the increment in Te(a_{12}) is less than Ti(a_1) + Ti(a_2). If two instance activities a_1 and a_2 are almost same size, and the resource provided for the merged activity a_{12} is almost same, it will be possible to do validation at the same timing as before. So, there will be no risk from the viewpoint of monitoring. But the risk of schedule delay will become bigger than before.

These are the basis of the economic discussions about class and instance activities in process reconfiguration. In the next chapter, we will present four fundamental patterns of process re-engineering and discuss about process re-configuration and its impact using these patterns.

Process Re-engineering Patterns

In this chapter, we present the four types of patterns useful for process re-engineering. They are (1) Simplifying, (2) BPR, (3) Pipeline, and (4) Spiral pattern. And the purpose, motivation, applicability, structure, and consequences of each pattern are described within each pattern.

These four patterns are different in their structures. There are various trade-offs caused from those structural differences. Following trade-off factors should be considered when we adopt these patterns for re-engineering:

(a) Turnaround Time

(b) Overhead Time (Tp, Tc, Ti)

(c) Easiness of process monitoring

(d) Maximum amount of resources needed

(e) Risk (Schedule delay etc.)

(f) Organization

The easiness of process monitoring is simply measured by the number of confirmation, namely the number of instance activities. In some cases, there may be environments or tools that enable to monitor process during its execution. But it may be a rather unusual situation. We treat the progress of an instance activity as a black box.

In our pattern descriptions, we use CSP-like notations as follows:

• i : A	Activity A with name i
• $A_1 ; A_2$	Normal execution of activity A_1 followed by A_2
• *A	Repetition of activity A
• $A_1 \| A_2$	Parallel execution of activity A_1 and A_2
• $A_1 \|\| A_2$	Activity A_1 and A_2 interleave

This provides the simple expression of the activity and the relations between the activities. And we also depict the process by graphical notation. In it a square shows an activity and a circle indicates the channel between activities, that is, an information that will be stored as a document.

1.4 PRP 1 - Simplifying Pattern

This is the most fundamental pattern of process re-engineering. It removes a particular activity from a process.

Purpose

Minimize overhead in a process by removing an activity whose role is degenerated.

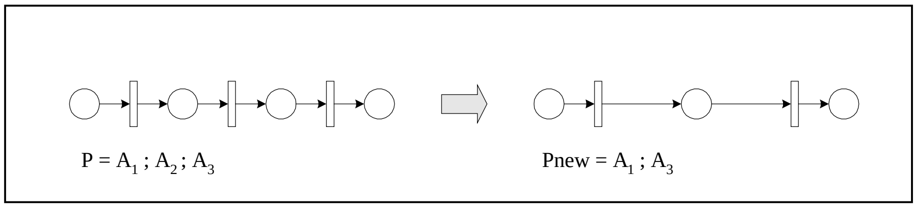
Motivation

Reorganizing the pre-defined process, when it becomes out-of-date because of the change in the adopted development methodology or product characteristics.

Applicability

- When the actual process looks different from the given standard process.
- When it is possible to change the process by switching human agents around.

Structure



Remove unnecessary activity A_2, or merge it into A_1 or A_3.

Consequences

(a) Turnaround time: shorter

(b) Overhead: less

(c) Easiness of monitoring: same (or less)

(d) Maximum amount of resource: same (or more)

(e) Risk: same (or more)

(f) Organization: same (or partially co-exist)

Related Pattern

In the BPR pattern, multiple activities are merged into single activity.

1.5 PRP 2: BPR Pattern

Merging multiple process elements and make them to be executed concurrently. This pattern is adopted in BPR (Business Process Re-engineering).

Purpose

Decrease process execution time by the partition of the product and unified team activities.

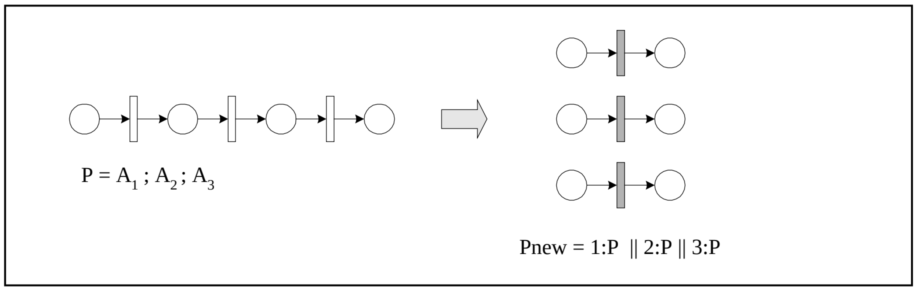
Motivation

When a process is decomposed into many fine-grained activities, it is difficult to manage the interaction between those activities. In such situation, many agents with different roles participate in the process, and it is difficult to maintain unified process goal. If it becomes necessary to modify the process goal, each activity can not adapt them for the change. If the interrelationship between multiple products delivered from the process is not so strong, or there are no cross relationships between them, the process can be divided product-wise, and each product can be produced by a number of parallel sub-processes executed by different small group of agents.

Applicability

- Software product can be divided into units with almost no interrelationship.
- Each sub-process is executed small group of human agents with a specific goal. It is necessary that there is a possibility of sharing common knowledge and experiences within the group.

Structure



Consequences

(a) Turnaround time: shorter

(b) Overhead: less

(c) Easiness of monitoring: uncertain

(d) Maximum amount of resource: same

(e) Risk: same

(f) Organization: large scale change

Usually, each activity has a single or small number of goals. In this BPR type pattern, all of the roles (agents) participate with a single activity. As a result, communication between agents increases. So, they should be organized as a small physically closed group.

Related Pattern

The spiral pattern has the same goal of increasing turnaround time. But the spiral pattern is used to develop products incrementally, on the other hand, BPR pattern’s main goal is just to increase turnaround time.

1.6 PRP 3: Pipeline Pattern

Purpose

Shorten process execution time introducing parallelism.

Motivation

Parallel execution of the activities is the most effective way to increase turnaround time. But, the introduction of the parallelism is not so simple because there are some interdependency between activities.

Applicability

This patter is applicable when there is a basic system and introduce a kind of change all over the system components. An example is the case of applying specific changes all over the subsystems within a maintenance process. More typical example is the case where a same agent can not execute needed for a product continuously because each activity needs specific expertise: for example, one person develops the functional specifications for all products, and the other agent will defines user interfaces of all products.

When activities interleave like:

$$A_1 = A_{11} \mid\mid\mid A_{12} \mid\mid\mid A_{13}$$

$$A_2 = A_{21} \mid\mid\mid A_{22} \mid\mid\mid A_{23}$$

$$A_3 = A_{31} \mid\mid\mid A_{32} \mid\mid\mid A_{33}$$

a pipeline can be configured as follows:

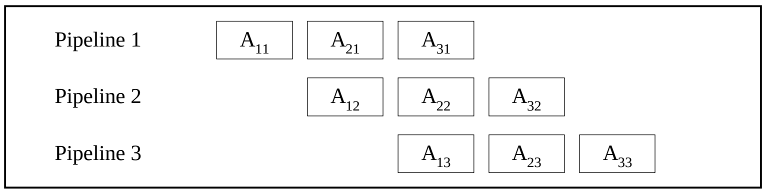
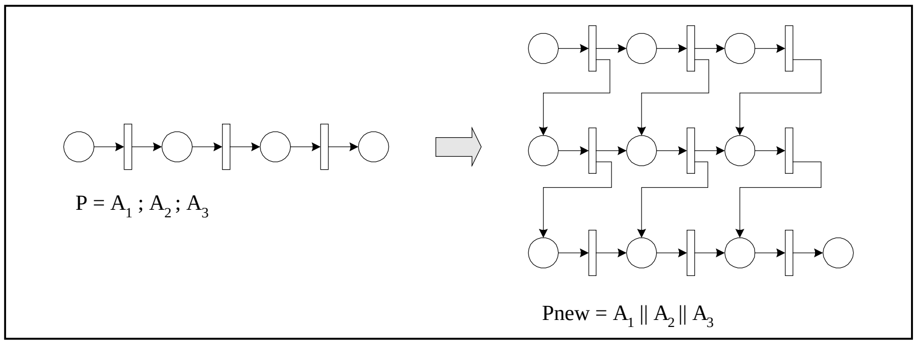


Figure 2. The Execution of Pipeline

In this case, the average amount of resources: 9/5 = 1.8, and the cycle time becomes 5/9 = 0.56.

Structure



Consequences

(a) Turnaround time: shorter

(b) Overhead: same

(c) Easiness of monitoring: same

(d) Maximum amount of resource: more

(e) Risk: same

(f) Organization: To be changed

In principle, each specific agent executes each activity. Then, to minimize communication error, it is necessary to implement an organizational change to assure that each activity can be assigned to a specific agent (e.g. an agent only participates execution of A_{11}, A_{12}, and A_{13}).

Related Pattern

It is same as spiral type in the sense of repetitive application of activities of the same class. But the repetition is not concurrent. It occurs different point on the time axis.

Misc.

As we stated earlier, it is difficult to introduce parallelism without using pipeline type patterns. But, in the case of some development methodology, it is partially possible. In Shlaer-Mellor's Recursive Development, a method is proposed to perform application analysis and decision of the implementation mechanism concurrently after the decomposition of the target problem [2].

1.7 PRP 4: Spiral Pattern

Execute development activities like analysis-design-test repetitively.

Purpose

- Decrease risks.
- Increase user satisfaction.

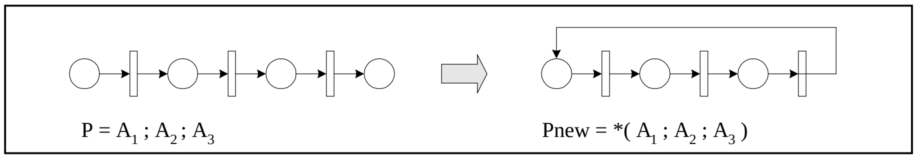
Motivation

In the discussion of software lifecycle, there are many spiral patterns. The most famous one is the spiral model of software lifecycle [3]. Major motivation behind this lifecycle model was early validation. Users wanted to validate the behavior or appearance of the software system earlier. And developers wanted to confirm the feasibility of implementation. Also, managers wanted to have more reliable schedules in their hands. To achieve these goals, cyclic development of prototypes and risk analysis is performed. This model was invented to overcome the fatal defect of the classic waterfall model, where users can not see the shape of product until final building phase is completed.

Applicability

This pattern is applicable when you can get the full appreciation from users.

Structure



$$P = *(A_1 ; A_2 ; A_3) \text{ or } P = A_1; *(A_2 ; A_3) \text{ or } P = A_1 ; A_2 ; *A_3$$

There are several different ways to introduce repetition. Total sequence of activities can be repeated, or some partial sequence of activities can be repeated.

Consequences

(a) Turnaround time: same or longer

(b) Overhead: more

(c) Easiness of monitoring: more

(d) Maximum amount of resource: more

(e) Risk: less

(f) Organization: same

This pattern does not contribute to the shortening of the process execution time, but the development schedule become more reliable. Total amount of the effort will be increased, but the quality of the products will be improved.

Related Pattern

In the case of $P = *(A_1; A_2; A_3)$, if A_1 is repeated before the completion of A_3, this pattern is same as pipeline.

Related Studies

There have been a number of studies using the term *process patterns*. Ambler defines process pattern as "a collection of general techniques, actions, and/or tasks (activities) for developing object-oriented software" [4]. He sorted the elements of object oriented software development process as task, step, and phase according to their granularity, and provided the desirable process. In Catalysis Approach, there is a clear distinction between development context and phase, and the methodology provides necessary process for each level [5]. In the case of OPEN, they do not use the term process pattern, but the execution procedure of each tasks are shown as patterns [6].

Common strategy in the above is as follows: It is impossible to solve all the problems with one single method and process. At first, break down the total problem solving process into the elements (phase in Catalyst, task in OPEN), provide effective methods for each process elements, and give a guideline how to use them in a combined way. For that purpose, the concept of process pattern is used. Similarly, Beedle defined a pattern language for the business process re-engineering [7].

PRPs described in this paper are purely oriented to the process structure. We have no concern about the contents of each process element. And it's also able to apply non-object-oriented processes. In that sense, our approach is more abstract. It is a meta-approach for the process re-engineering problem.

Summary

In this paper, we have discussed about the fundamental patterns in process re-engineering. Before describing these patterns, we clarified the distinction between process and activity, and also class/instance level in activity analysis.

Usual discussions of process scheduling are at the instance level, but our PRPs mainly deal with class-level activities.

In this paper, the emphasis was put on the structural analysis of the process to derive the possible patterns of change. But, the socio-psychological aspects of the process re-engineering are also important, because human agents execute most part of process. For example, in the case of BPR type pattern, people of different roles participate into a single activity. They can share a common goal for the product. It is better than the situation where the process is divided into a number of activities and not easy to establish total perspective, and the quality of the product will be improved naturally.

The concept of PRP proposed here is considered as a powerful tool to implement an optimized process (CMM level 5). To achieve that goal, it is needed to do more detailed study on the psychological aspects of process and to enrich PRPs with collecting a number of best practices.

References

1. Garg, P. K., et al., Process-centered software engineering environments, IEEE, 1996.

2. Shalaer, S., Mellor, S., "Recursive Design", Computer Language, March, 1990.

3. Boehm, B., "A Spiral Model of Software Development and Enhancement", ICSE, 1985.

4. Ambler, S., PROCESS PATTERNS, Cambridge University Press, 1988.

5. D'Souza D., F., et al., OBJECTS, COMPONENTS, AND FRAMEWORKS with UML, Addison-Wesley, 1998.

6. Graham, I., THE OPEN Process Specification, Addison-Wesley, 1997.

7. Beedle, M. A., "Pattern Based Reengineering", OBJECT magazine, Vol.6 No. 11, 1997.

Modeling Framework and Supporting System for Process Assessment Documents

Makoto Matsushita[1], Hajimu Iida[2], and Katsuro Inoue[1]
[1]Osaka University, Osaka, JAPAN
[2]Nara Institute of Science and Technology, Nara, JAPAN

Abstract

Process is usually assessed by comparing with the assessment documents, though the information for the assessments are scattered; it is a time-consuming job to gather them. In this paper, we investigated the method to obtain information for process assessment by means of SPICE (Software Process Improvement Capability dEtermination). First, we extracted the information related to processes, products, and levels. Then, to construct the SPICE documents based on that extracted information, SGML (Standard Generalized Markup Language) was adopted. Based on the constructed SPICE documents, we have made prototypes of two tools, a tool that shows the information about process assessment on the display, and a tool that investigates relations between processes and products. These tools enable us easily to get information for process assessment, and to improve process assessments greatly.

Introduction

Improving software development processes are the important issues to achieve effective software production or to reduce the cost of software development. To improve this, first we should evaluate what the target of software development process is going on.

Recently there are various studies of software process assessment and software quality assurance, and the fruits are widely used in software development organization [4,8]. There are lots of evaluation methods and reference model, including CMM (Capability Maturity Model) [6,7], of SEI, ISO-9000 series standards [14], SPICE (Software Process Improvement Capability dEtermination) [17], etc.

Software process is usually assessed with examining the documents of the projects or having interviews with the engineers and managers, by the experts of software process assessment standards. However, such procedure is a time-consuming job to execute, and the costs of this are very large; it seems that it is difficult to do [4,11].

In this paper, we have designed the simple model that does not introduce our original interpretation of software process assessment standards. The model consists of three elements and relationships between them. The model is described with SGML (Standard Generalized Markup Language) tag. We reorganized software process assessment documents as SGML documents. Using these documents, these standards can be easily handled formally and implemented easily. We have also designed a process assessment supporting system for self-assessment, and implemented two process assessment supporting tools.

SPICE

SPICE is one of software process assessment standards[19], and now it is standardized by ISO. The whole document of SPICE is about 400 or more pages. SPICE arranges the whole activities in software development environment into five “process categories”. Each process category consists of a set of “processes”, and process is consists of a set of “base practice”. SPICE has yet another axis of activities for evaluating the capability for each software development activity named “capability level”. Each capability level consists of a set of “common features” which represent the characteristics of activities. Each “common features” consists of a set of “generic practice” (fig.1) [21].

[19] Current version of SPICE was changed described in this paper. However, our proposed framework is independant from old SPICE specification; we think that adapting current SPICE to our framework should be possible.

The rough procedure to evaluate a software development process has three steps; first, it should be decided what to be evaluated. Then, information is gathered from the target process. Finally, information is evaluated checking with the standards to sum up the result [20].

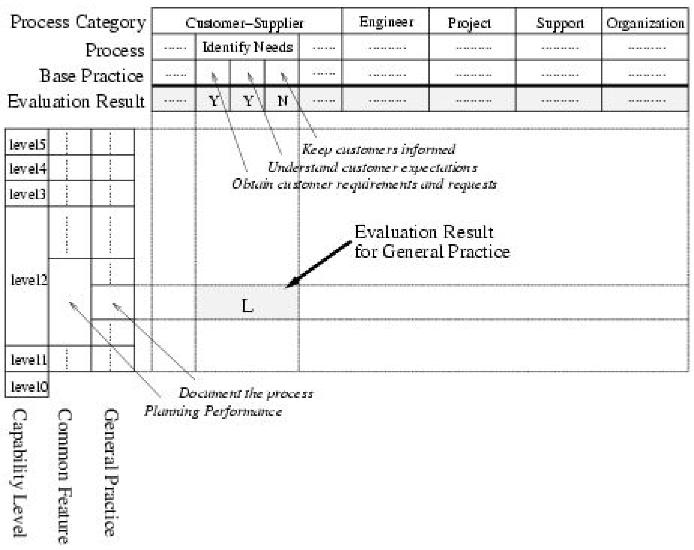


Figure 1: The classification of activity and the decision of assessment in SPICE framework

Generic Modeling for Software Process Assessment Standards

In this section, we show our modeling approach of generic software development process in the assessment standards with the SPICE framework.

1.1 Modeling Policy

The whole of software development activities focused in the software process assessment standards are modeled with following three types of elements and four types of relationships of these elements.

Elements

- Task: Task represents a set of activity in software development environments. Process categories, processes, base practices in SPICE will be candidates.

- Level: Level represents the achievements of software development work. Capability levels, common features, and generic practices in SPICE will be candidates.

- Product: Product represents generated materials and/or documents when tasks go on, or prepared when the tasks start.

Relationships

- Task – Product: Relationship about “a product is needed to enact a task”.

- Task – Task: Relationship about “an another task is performed to enact a task”.

- Level – Task: Relationship about “a task is performed to achieve a level”.

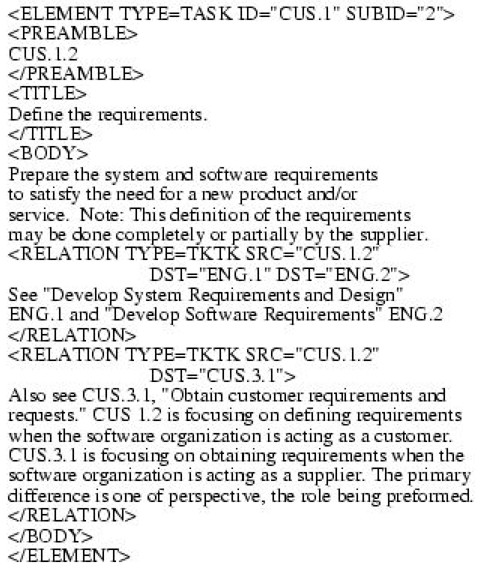
- Level – Product: Relationship about “a product is needed to achieve a level”.

The elements defined in our model can be extracted from the elements of SPICE standards, and the relationships can be extracted from the descriptions of SPICE standards. Actual description of our model is shown in section 3.2.

1.2 Model Description with SGML

In our approach, the model proposed in section 3.1 is described with SGML (Standard Generalized Markup Language) tags that are inserted into the original documents. In general, formed documents are structured as SGML documents; it enables to process documents, to build documents databases, to exchange document formats [10].

We define two tags to markup the documents, ELEMENT and RELATION for elements and relationships of our model, respectively. The information of each element and relationship is described as attributes of these tags, and it represents clearly the meanings written in software process assessment standards. We also define other tags to represent the structure of document itself, including the preamble, a name of base practice, etc. Figure 2 shows a fragment of document.



```
<ELEMENT TYPE=TASK ID="CUS.1" SUBID="2">
<PREAMBLE>
CUS.1.2
</PREAMBLE>
<TITLE>
Define the requirements.
</TITLE>
<BODY>
Prepare the system and software requirements
to satisfy the need for a new product and/or
service.  Note: This definition of the requirements
may be done completely or partially by the supplier.
<RELATION TYPE=TKTK SRC="CUS.1.2"
                    DST="ENG.1" DST="ENG.2">
See "Develop System Requirements and Design"
ENG.1 and "Develop Software Requirements" ENG.2
</RELATION>
<RELATION TYPE=TKTK SRC="CUS.1.2"
                    DST="CUS.3.1">
Also see CUS.3.1, "Obtain customer requirements and
requests." CUS 1.2 is focusing on defining requirements
when the software organization is acting as a customer.
CUS.3.1 is focusing on obtaining requirements when the
software organization is acting as a supplier. The primary
difference is one of perspective, the role being preformed.
</RELATION>
</BODY>
</ELEMENT>
```

Figure 2: An example of a SPICE document with SGML

Supporting System

This section describes an experimental supporting system for software process assessment. The system employs SPICE as an assessment standard, and uses tagged documents, which describes in the previous section. The purpose of this system is to evaluate own software development process by developers or managers themselves.

The system supports to view the documents, find a relationship of documents, to apply logical operation, to maintain correspondences between the standard and actual software process, to show assessment information, and to register/accumulate assessment results.

The system consists of two tools and associated database (figure 3). There are two tools; a tool that investigates the information about process assessment document itself, elements and relationships written in the documents, and a tool which manages the result of software process assessment. Each tool uses SPICE documents that is tagged based on our model, and saves the analyzed result to the database. The result of software process assessment is also saved to the another database.

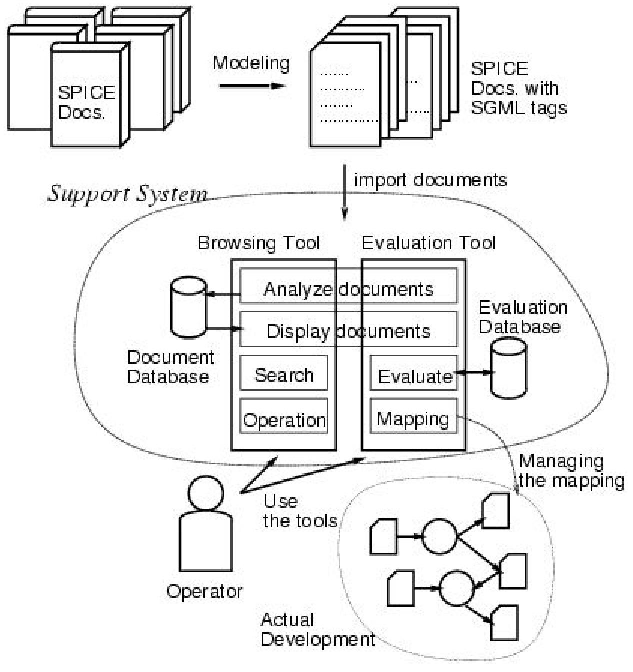


Figure 3: System overview

1.3 Document Viewer

The document viewer shows the information described in SPICE documents, the graphs of reference between tasks etc. Figure 4 shows a screen-shot of the document viewer. This tool has following features:

- Word searching: The tool shows a fragment of documents corresponding to a user's input.

- Keyword searching: The tool enumerates the name of task and/or products corresponding to a user's input. In addition, corresponding documents are shown by selecting enumerated one.

- Relation searching: The tool traverses the relationships in the documents. Traversing may be recursive, so derivative relations can be found.

- Logical operation: Operations described above can be combined each other by logical operations, such as getting intersection of two search results. Operation results can be *piped* to other operation's input.

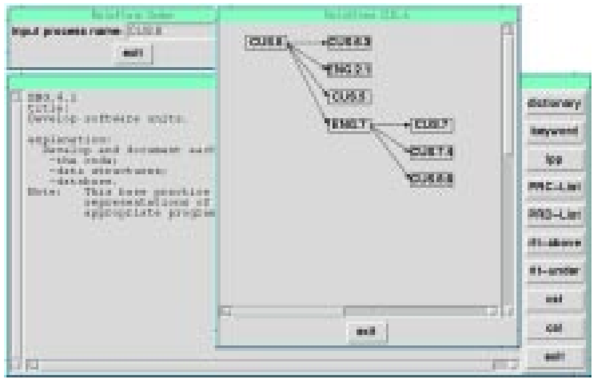


Figure 4: The document viewer

The document viewer shows the result of these features visually. If the result will be a set, the tool shows the result as a list; if the result will be a tree structure, the tool shows the graph of the tree. In figure 4, the result of tasks that are referred from a task is shown as tree structure.

1.4 Evaluation Supporting Tool

The evaluation-supporting tool is for self-assessments of software process. Figure 5 shows a screen-shot of this tool. This tool has following features:

- Document viewer: It is a subset of document viewer tool previously shown. This tool shows a definition of words, task or products. If tasks are already evaluated, its results are also shown.

- Database for mapping the standards and an actual environment: The tool manages the relation between elements described in assessment documents and actual software development environment. These relations are sorted and showed in a table, and stored into a database. The files registered to database can be displayed.

- Database of assessment result: Users may enter the result of the assessment; specify a task, or a task and associated level, then enter the evaluation (two-level or four-level evaluation) with a dialog. Evaluation result can be selected with a button. These results are stored into a database.

- Collecting assessment results: The tool collects the result of evaluation, then sums up each process category with a table. In addition, the tool sums up each capability level and process category, then shows the results with their capability maturity percentages.

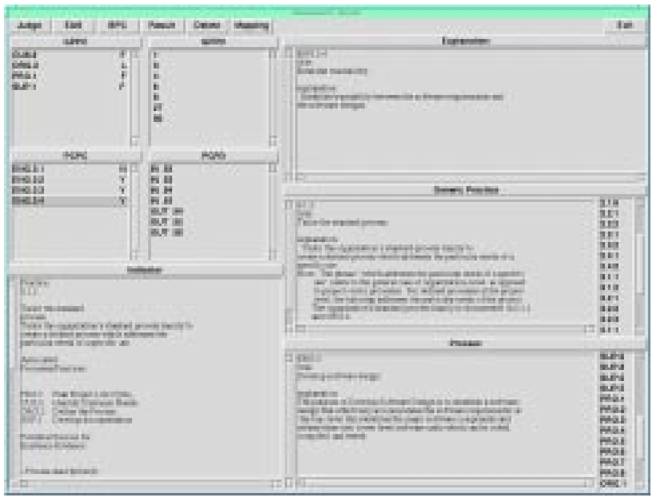


Figure 5: The evaluation supporting tool

We have implemented and evaluated a prototype of this system with a software development environment based on ISPW-6 [2].

Discussion

1.5 Modeling Approach

Our modeling approach described in section 3.1 can retrieve the information described in a large software process assessment documents such as SPICE. In general, designing an assessment supporting system requires own interpretation or explanation that is pre-defined by tool designer, then implements a system with this requirement [3]. This approach may include a wrong interpretation of assessment document to the system design, and may lead to a wrong result of assessment. Our approach uses the original definition or relationships written in the original document; it is less different from the original interpretation of assessment.

1.6 Tagged Document

In this work, we employ SGML as a description language of our model, design and implement a tool based on tagged documents. SGML is commonly used for various objectives such as reusing document or full-text database [5,9]. However, these applications are intended to handle many files in the same format. Our system handles large single file and provides a feature to operate a file. In addition, there are many SGML-based documenting support environments [12,15,16], and they become popular tools for electric documents. However, these environments have so many features and we want to keep the whole system to be simple.

Recently, WWW (World-Wide Web) and HTML (HyperText Markup Language [1]) are widely used in many places, especially in the Internet. There are many tools or environments for HTML and we may use these powerful tools. However, our purpose requires our own tag definition and implementation and it requires some extension to HTML format; it should lose the portability of HTML, so it should be avoided. We are now investigating an XML as our model representation format.

1.7 Supporting System

Our system is for the self-assessment, to support to give a hint of the process improvement. Such assessment methods are not verified and guaranteed by external organization, however, it is easy to execute with appropriate assessment criterion; software assessment activities are widely used in lots of software development organization.

It is a long term to proceed a software process assessment to an actual software development project. Our tools has evaluation results database, so that suspending and resuming the evaluation procedure is quite easy; it can be used for long term evaluation. Our tools also support a database for relation of assessment standards and actual environment, so it may supports the decision of evaluation.

Our current prototype supports two types of activities in software process assessment, gathering an information for evaluation, and calculation of the evaluation result. These features are designed and implemented with the procedures defined in an assessment standards [20], so users can do the right way to assess a target process. Software process improvement based on the assessment result will bring about better and effective software development. This prototype currently depends on SPICE standards. However, our model does not depend on it; adapting other standards to the prototype is possible.

Conclusion

We have proposed a model of software process assessment documents, and defined it as SGML documents. We also designed and implemented a prototype of software assessment supporting system. Using our approach and system, software assessment documents can be formalized easily and the result of system brings simple software assessment method.

As a further work, validation of our model and environment through experiments and applying other software process assessment standards to our model are planned. In addition, supporting other phases in process evaluation (preparation, improvement planning, etc.) is needed to our system.

References

[1] Berners-Lee, T. and Connolly, D.W., “Hypertext Markup Language - 2.0”, RFC1866, Massachusetts Institute of Technology Laboratory for Computer Science / The World Wide Web Consortium, ftp://ds.internic.net/rfc/rfc1866.txt, 1995.

[2] Kellner, M.I., Feiler, P.H., Finkelstein, A, Katayama, T., Osterweil, L.J., Penado, M.H. and Rombach, H.D., “Software Process Modeling Example Problem”, In Proceedings of the 6th Int. Software Process Workshop, pp.19-29, 1990.

[3] Omoto, N., Komiyama, T. and Fujino, K., “Software Process Assessment Support System SPATS”, IPSJ Technical Journal, 95-SE-102-28, pp.159--164, 1995.

[4] MacLennan, F. and Ostrolenk, G., “The SPICE Trials: Validating the Framework”, In Proceedings of the 2nd International SPICE Symposium, pp.109--118, 1995.

[5] Morita, U., Suzuki, M., Miyagawa, K. and Hamanaka, H., “A Trial For Development of DTD for "JOHO KANRI" and "JOHO KANRI" Full Text Database by Using HTML”, IPSJ Technical Report, 95-FI-37-2, pp.7--14, 1995.

[6] M. Paulk, B. Curtis, M. Chrissis, and C. Wever, “Capability Maturity Model for Software, Version 1.1”, Software Engineering Institute, CMU/SEI-93-TR-24, 1993.

[7] M. Paulk, B. Curtis, M. Chrissis, and C. Wever, “Key Practices of the Capability Maturity Model, Version 1.1”, Software Engineering Institute, CMU/SEI-93-TR-25, 1993.

[8] H. Saiedian, and R. Kuzara, “SEI Capability Maturity Model's Impact on Contractors”, IEEE Computer, Vol.28, No.1, pp.16--26, 1995.

[9] Takayanagi, Y., Sakata, H. and Tanaka, Y., “Full-text Database System Based on SGML”, IPSJ Technical Report, 93-CH-18-5, pp.35--42, 1993.

[10] Tanaka, Y., “Standardization of SGML”, Journal of IPSJ, Vol.32, No.10, pp.1118--1125, 1991.

[11] I. Woodman, and R. Hunter, “Analysis of Assessment Data from Phase 1 of the SPICE trials”, Software Process Newsletter, No.6, pp.1--6, 1996.

[12] DocIntegra, http://www.hitachi.co.jp/Prod/comp/soft1/open/docint.htm, Hitachi Ltd., 1995.

[13] ISO 8879, “Information Processing - Text and Office System - Standard Generalized Markup Language (SGML)”, 1986.

[14] ISO 9000-3 Guidelines for the Application of ISO 9001 to the Development, Supply, and Maintenance of Software, 1991.

[15] OLIAS, http://www.fujitsu.co.jp/hypertext/granpower/topics/olias/olias.html, Fujitsu Limited, 1996.

[16] Panorama Pro, http://www.sq.com/products/panorama/panor-fe.htm, SoftQuad Inc., 1996.

[17] The SPICE Project, “Software Process Assessment -- Part 1: Concepts and Introductory Guide”, Version 0.02, 1994.

[18] The SPICE Project, “Software Process Assessment -- Part 2: A Model for Process Management”, Version 0.01, 1994.

[19] The SPICE Project, “Software Process Assessment -- Part 3: Rating Process”, Version 0.01, 1994.

[20] The SPICE Project, “Software Process Assessment -- Part 4: Guide to conducting assessments”, Version 0.02, 1994.

[21] The SPICE Project, “Software Process Assessment -- Part 5: Construction, Selection and Use of Assessment Instruments and Tools”, Version 0.02, 1994.

An architecture for defining the processes of the software and systems life cycles

Terence P. Rout and Peter Bernus

Software Quality Institute and

School of Computing and Information Technology

Griffith University

Queensland, Australia

Abstract

Attempts to define consistent reference models for systems and software life cycle processes have been hindered by the lack of an acceptable underlying architecture for such models. The use of existing enterprise architecture models is put forward as an appropriate solution to this problem. By assigning processes to identifiable entities in the enterprise model, problems arising from the lack of definition of scope and of the relationship between different processes can be resolved. The use of appropriate enterprise models as the basis for process reference models is proposed as a resolution of the problems associated with the development and application of process reference models for the software engineering domain.

Introduction

Over the past five years there have been several attempts to define a set of standard definitions for the processes of the software and systems life cycle. These attempts have been hindered by lack of consensus over the structure and content of an appropriate reference model, but substantial progress has been made. This paper addresses one of the key remaining unresolved issues in this task.

Background

The first comprehensive set of definitions of software life cycle processes was contained within ISO 12207 - Software Life Cycle Processes [1]. During the development of ISO 12207, the SPICE Project developed a set of definitions in the form of a Baseline Practices Guide [2] having a different (though similar) architecture to that of ISO 12207; over the next few years, this model was modified and brought into strong alignment with ISO 12207, and finally approved as ISO/IEC TR 15504-2, a “Reference model for processes and process capability” [3].

The model originally defined in ISO 12207, while promising, had a number of deficiencies which detracted from its wide application. Principal among these was the fact that the standard was developed from the perspective of process implementation, rather than definition. The process descriptions in 12207 are based around prescriptions of tasks and activities that need to be performed in order to implement the process, and the intent of the process is described only in terms of the performance of these activities. The Baseline Practices Guide has a similar problem, though it included statements of basic purpose of each process in the model. In ISO/IEC 15504, there was the emergence of a consensus - at least within the standards community - as to how the standard descriptions of processes should be formulated: processes are defined by describing the purpose and the outcomes of successful implementation. There is also general agreement on the nature of these descriptions; for example, ideally, each outcome would have the following characteristics:

- capability-neutral (“capability” is used here in the sense of 15504)
- role-independent
- succinct
- not a restatement or functional decomposition of the purpose
- phrased in terms of a continuing responsibility to:

  1. produce and maintain an artefact;

2. achieve and maintain a state; or
3. meet a constraint.

This approach leads to generally acceptable descriptions of processes of the software life cycle, and early experience indicates these are also applicable to other domains, including that of systems engineering. The development of ISO/IEC 15288 - Systems life cycle processes [4,5], however, has highlighted the lack of consensus on another critical aspect of this debate - the need for a common architecture for the process models. The lack of a common understanding of this issue is so pervading that a current proposal for resolution avoids the issue totally, putting forward a view that would result in a "repository" of process descriptions rather than a true reference model.

The need for a common process architecture

A common architecture for life cycle process models is needed if the models thus defined are to have truly universal application, and are to avoid problems of incompatibility and interpretation between different uses and domains of application. The architecture shows the relationships between processes, and describes clearly how a given process contributes to the relevant process life cycle.

The initial architecture proposed was that of ISO 12207, which established three classes of processes: Primary, Supporting and Organizational. The Primary processes included all of the product life cycle phases for software, together with processes for Supply and Acquisition. The Baseline Practices Guide proposed five Categories of processes: Customer-Supplier, Engineering, Project, Support and Organization. The five categories were retained but extensively revised for ISO/IEC 15504; they are now named Customer-Supplier, Engineering, Support, Management and Organization. The processes are also now strongly mapped to the architecture of ISO 12207 - the first two categories comprise the Primary Processes, while the last two comprise the Organizational Processes. Figure 1 shows the processes of ISO/IEC 15504-2, and their relationship to the classes and processes of ISO 12207.

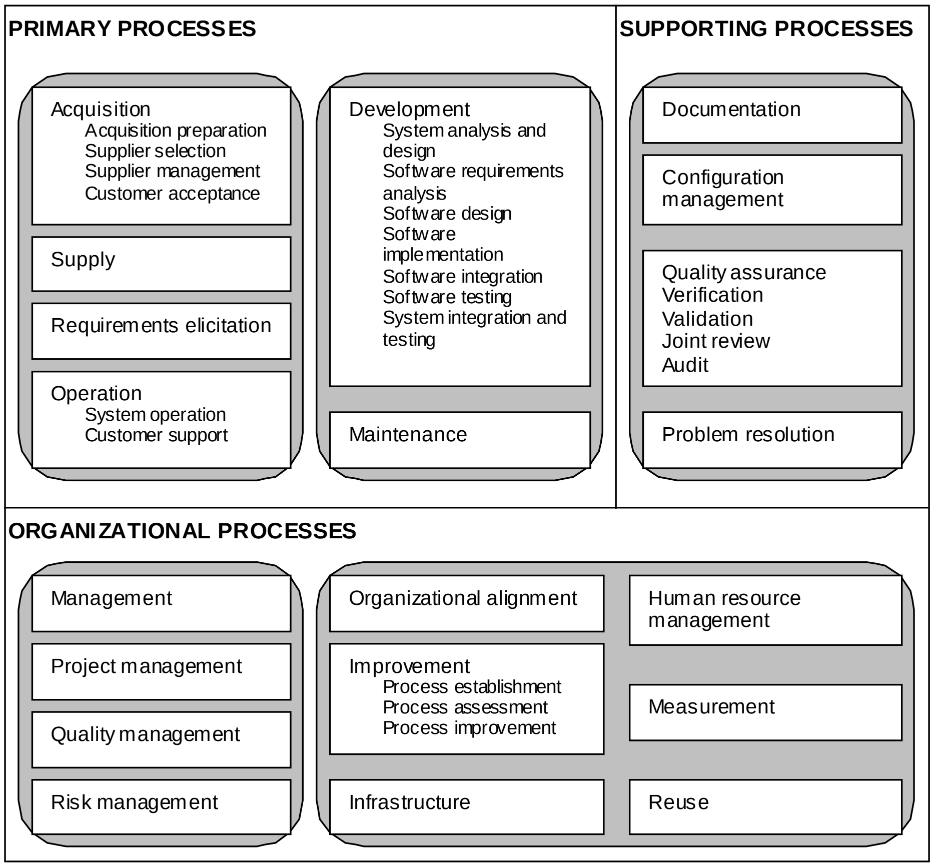


Figure 1 - Processes defined in ISO/IEC 15504-2

With the System Life Cycle standard, however, a considerably different architecture has been adopted (Figure 2).

There are obvious differences in the granularity of processes in the two models - for example, all of the “supporting processes” in ISO 15504 are subsumed into “activities” in the Assessment and Control processes in this model. There are also differences in the relationships between the different processes; ISO 15504 implies a relationship between Project Management, Quality Management and Risk Management (for example) that is absent in the draft for ISO 15288. While a simple mapping between the two models is obviously possible, there is little possibility of full-scale integration, without major change in accepted views.

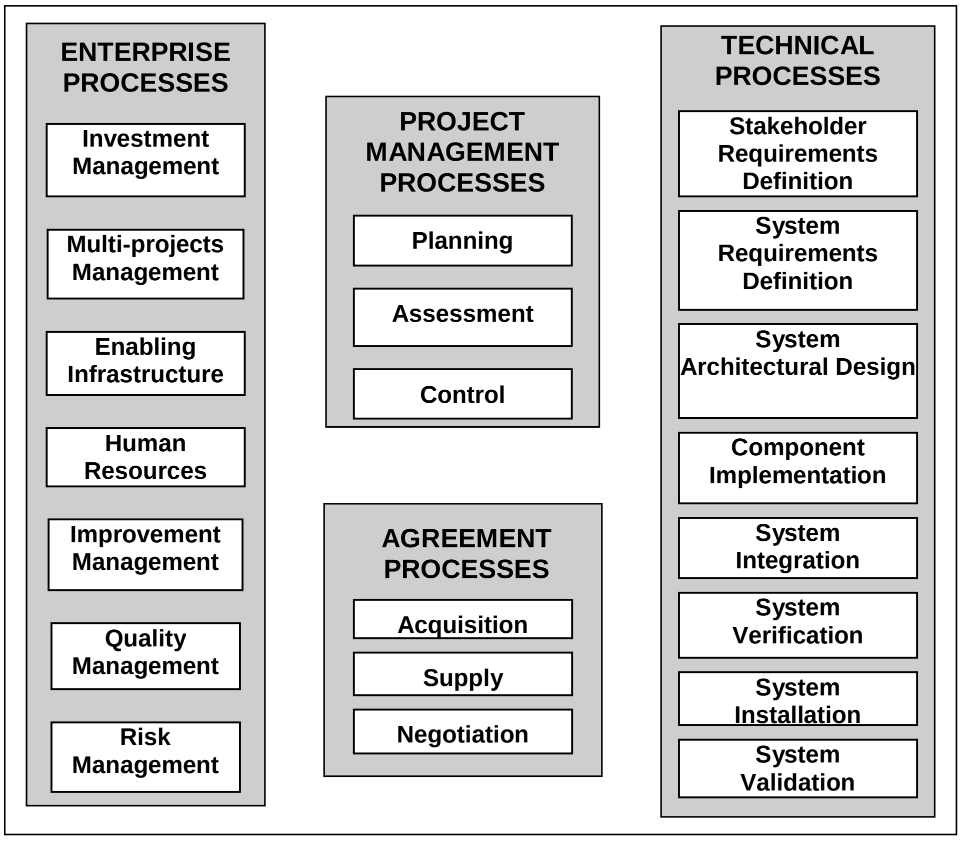


Figure 2 - Proposed Processes for the Systems Life Cycle

Part of the problem is that there is no recognised basis for the definition of a suitable architecture, beyond the views of the domain experts that specific groups of processes are in some way related.

Enterprise Models and Life Cycle Processes

A possible solution to the conflict is to adopt an architecture based upon an existing framework for modelling that has shown its value and relevance in spheres beyond that of systems and software engineering. One strong candidate for a suitable framework is in the field of enterprise modelling, where a generic architecture for modelling enterprises and related entities exists and has been extensively evaluated.

GERAM - an overview

GERAM (Generalised Enterprise Reference Architecture and Methodology) [5] defines a tool-kit of concepts for designing and maintaining enterprises and their products for their entire life-history. GERAM is meant to organise existing enterprise integration knowledge. The GERAM framework unifies two distinct approaches of enterprise integration, those based on product models and those based on business process design.

The framework has the potential for application to all types of enterprise entities meaning all entities the life of which is worthy of consideration in connection with the enterprise. Several applications exist in the industry to date. The framework organises methods, models and tools needed to build and maintain the integrated enterprise [6] and its products. GERAM is the basis for ISO 15704-Requirements for enterprise-reference architectures and methodologies[7].

The GERAM framework identifies, in its most important component GERA (Generalised Enterprise Reference Architecture), the basic concepts to be used in enterprise engineering and integration (for example, enterprise entities, life-cycles and life histories of enterprise entities). GERAM distinguishes between the methodologies for enterprise engineering (EEMs) and the modelling languages (EMLs) which are used by the methodologies to describe and model, the structure, content and behaviour of the enterprise entities in question. These languages will enable the modelling of the human part in the enterprise operation as well as the parts of business processes and their supporting technologies. The modelling process produces enterprise models (EMs) which represent all or part of the enterprise operations, including its manufacturing or service tasks, its organisation and management, and its control and information systems. These models can be used to guide the implementation of the operational system of the enterprise (EOSs) as well as to improve the ability of the enterprise to evaluate operational or organisational alternatives (for example, by simulation), and thereby enhance its current and future performance.

The methodology and the languages used for enterprise modelling are supported by enterprise engineering tools (EETs). The semantics of the modelling languages may be defined by ontologies, meta models and glossaries which are collectively called generic enterprise modelling concepts (GEMCs). The modelling process is enhanced by using partial models (PEMs) which are reusable models of human roles, processes and technologies.

The operational use of enterprise models is supported by specific modules (EMOs) which provide prefabricated products like human skill profiles for specific professions, common business procedures (e.g. banking and tax rules) or IT infrastructure services, or any other product which can be used as a component in the implementation of the operational system (EOSs).

A key component of GERAM is GERA, the Generic Enterprise Reference Architecture, which defines the generic concepts recommended for use in enterprise engineering and integration projects. These concepts can be classified as human oriented; process oriented; or technology oriented concepts.

The process-oriented concepts defined in GERA are: *enterprise entity life-cycle* and *life-cycle phases*; *life history*; *enterprise entity types*; and *enterprise modelling* with *integrated model representation* and *model views*. Life-cycle activities encompass all activities from inception to decommissioning (or end of life) of the enterprise or entity. The different life-cycle phases define types of activities which are pertinent during the life of the entity. As one enterprise entity operates, it implements life-cycle functions of another entity - eg. as an engineering project operates, it may support the design and implementation activities of the product life-cycle. The different classes of entity recognised in GERA include:

Repetitive Service- and Manufacturing Enterprise Entity which are enterprises supporting one or more types or a families of products, produced in a repetitive or sustained mode. Examples are service enterprises, manufacturing plants, engineering firms, infrastructure enterprises, etc.

The products of the repetitive enterprise may be diverse, ordinary products or products which are enterprises themselves, e.g. a plant, or a project.

Project Enterprise Entity (often with a short life history) created for the one-off production of another enterprise entity. (E.g. one of a kind manufacturing projects, engineering projects, etc.)

The project enterprise are normally created by repetitive service and manufacturing enterprises. and are closely linked with the life-cycle of a single product or service. The products of project enterprises may be diverse, such as large equipment, buildings, systems etc., or an enterprise in its own right (e.g. a plant, or an infrastructure enterprise).

Product Entity - a very large class of entities including any artificial product, such as customer goods, services, hardware equipment, computer software, etc. These entities are not enterprises themselves, but their life-cycles are described by GERAM.

GERA provides an analysis and modelling framework which is based on the life-cycle concept and identifies three dimensions for defining the scope and content of enterprise modelling:

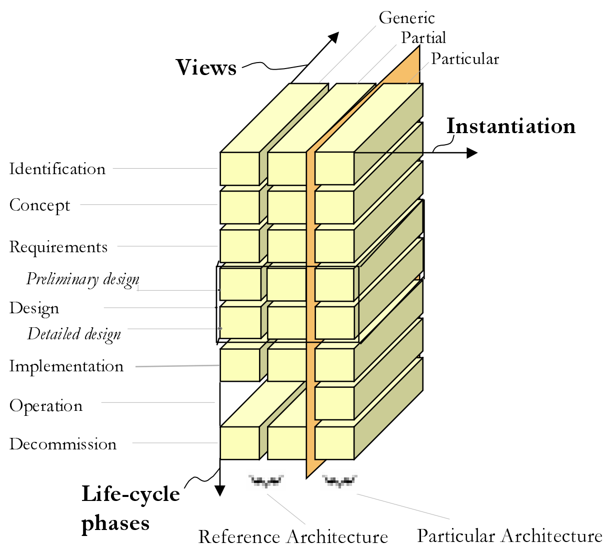


Figure 3 - GERA Modelling Framework

Life-Cycle Dimension: providing for the controlled modelling process of enterprise entities according to the life-cycle activities.

Genericity Dimension: providing for the controlled particularisation (instantiation) process from generic and partial to particular.

View Dimension: providing for the controlled visualisation of specific views of the enterprise entity.

Figure 3 shows the three dimensional structure identified above which represents this modelling framework.

Within this framework, process modelling is the activity that results in various models of the management & control as well as the service and production *processes*, and their relationships to the resources, organisation, products etc. of the enterprise.

Process modelling allows us to represent the operation of enterprise entities and entity types in all their aspects: functional, behaviour, information, resources and organisation. This provides for operational use of the models for decision support by evaluating operational alternatives and for model driven operation control and monitoring. It is this feature of GERAM that establishes its suitability for supporting the process reference models for the software and systems life cycles.

An enterprise-based process model

The nature of the enterprise-related entities associated with an undertaking - in this instance, a software engineering project - are shown in Figure 4. The architecture of the processes associated with an undertaking of this type is determined by associating the processes each to their relevant entity. Thus, the basic processes of the software product life cycle - from Software Requirements Analysis through to Software Operation - are associated with the product entity. The processes classed in ISO 12207 and ISO 15504 as "supporting" are associated with the Project entity, while other, organization-wide processes can be associated with the "Company" entity.

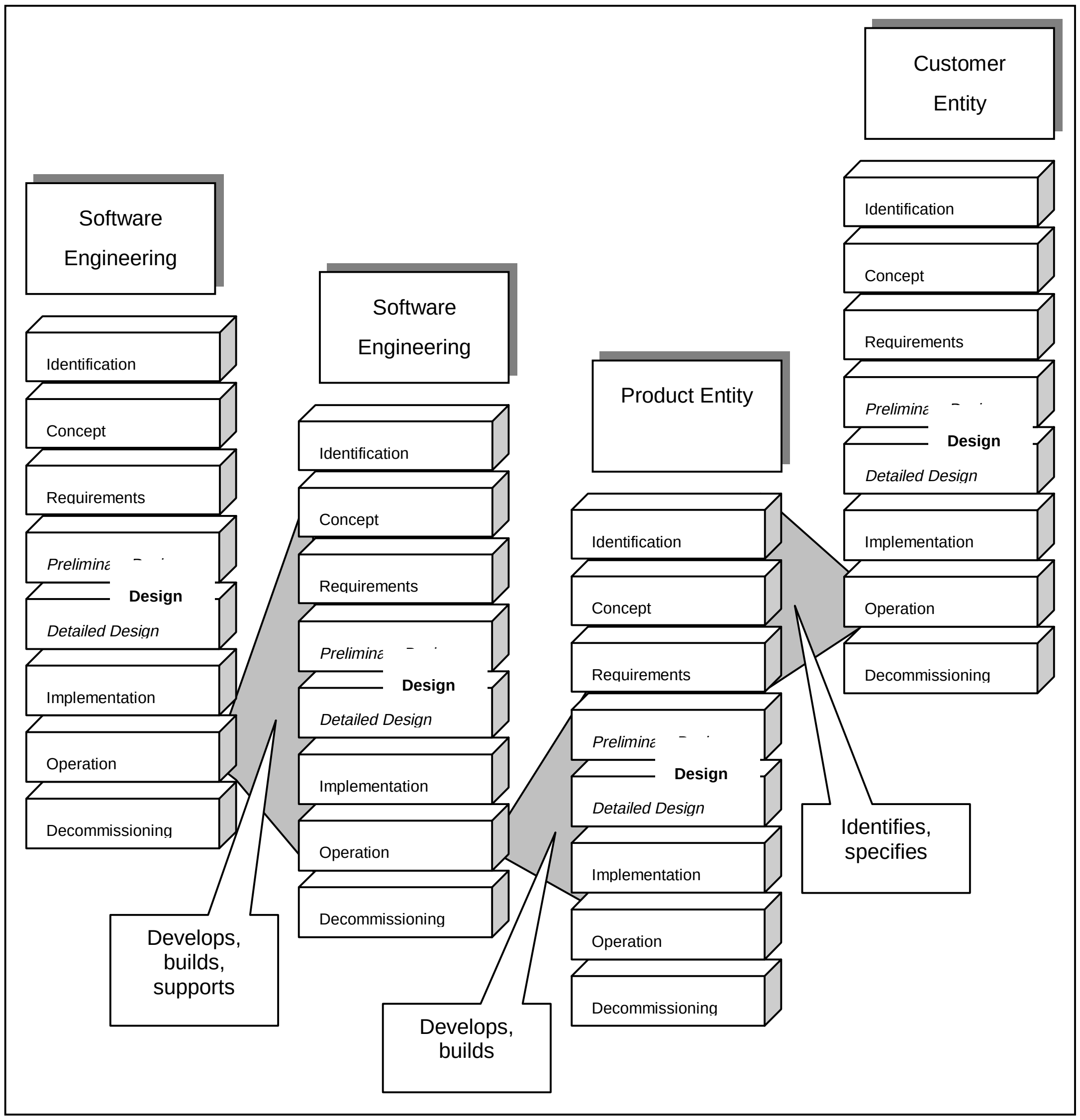


Figure 4 - Enterprise-based model for the software life cycle

The example in Figure 4 shows a relatively small sample of what would, if fully described, be a very complex architecture. Some of the strengths of this approach can be seen if the issues of acquisition and supply of the “product” entity are considered; Figure 5 displays some of the entities that would be associated with these functions, and show a possible classification of life cycle processes consistent with the architecture. In this expanded model, enterprise entities are identified for the Supplier; the Acquirer; and the Developer.

Separate entities can be identified for the various projects - the Acquisition Project, Supply Project, and Development Project - with interactions between the various primary entities.

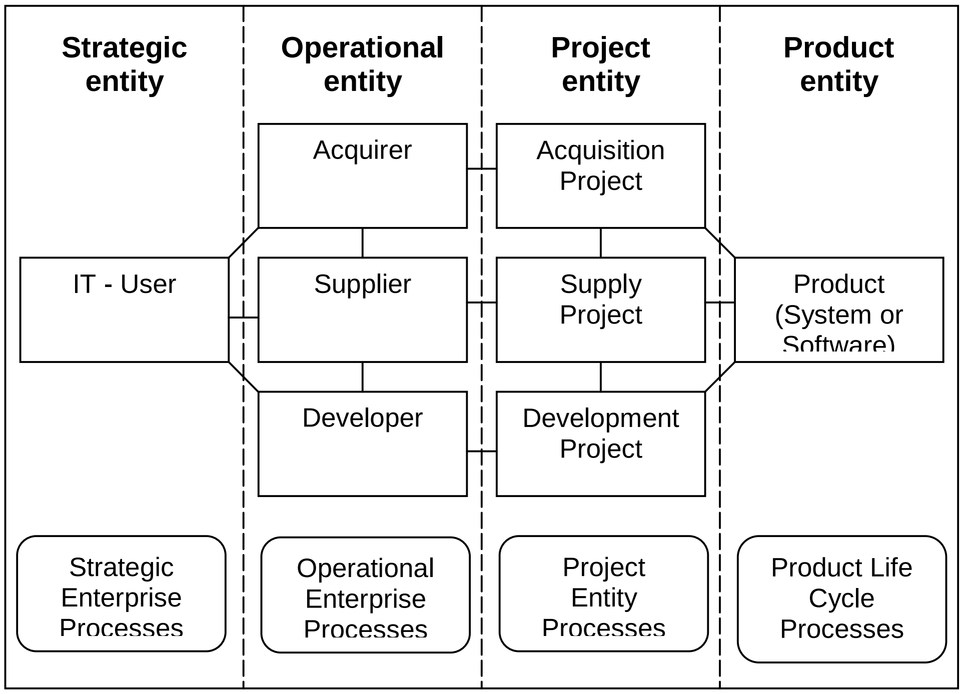


Figure 5 - Enterprise Entities and Life Cycle Processes

Each of the Project Entities is associated with the Product in different ways. The model shows how a common set of processes can be associated with all of these various entities, enabling a common understanding of the operations required to achieve the joint undertaking to be shared between all of the parties involved.

We are not attempting in this paper to describe a complete architecture for the set of life cycle processes; our intent is to demonstrate the value of an architecture based upon existing enterprise architecture models, and to perhaps lay the foundations for more detailed work to fully enunciate such a model.

Benefits of the proposed architecture

In examining the benefits of adopting a process model such as outlined above, it is necessary to first explore the reasons for the development and adoption of reference models for the life cycle processes. The primary justification for such a model is that of establishing a common vocabulary and understanding of the processes associated with the product life cycle. The existence and widespread acceptance of a set of standard descriptions of processes associated with a specified domain - be it software engineering, systems engineering, or (for that matter) conventional manufacturing is that it provides all of the stakeholders in an undertaking with a common vocabulary for expressing their understanding of the processes to be employed and controlled to achieve the goals of the undertaking.

The use of an established architecture as a framework for the process descriptions provides several benefits. In developing the reference model itself, the architecture provides structure and helps to “anchor” the definitions against accepted entities. The architecture helps also to provide a scope within which the process is intended to function; the lack of an evident scope for existing definitions can cause difficulties in interpreting the models.

In the application and use of the reference model, the existence of a well-structured architecture also has demonstrable benefits. It permits, for example, the specification of an appropriate set of implementable processes for a virtual enterprise - a relevant context for systems engineering. It also provides a bridge between the modelling concepts and techniques applied in integrated manufacturing approaches, and the sophisticated (and primarily personnel-based) life cycle concepts used in software engineering.

Summary and conclusions

There are inconsistencies and unresolved problems in the existing and proposed reference models for systems and software life cycle processes. In order to provide a firm foundation for integrated approaches to systems and software engineering, it is important that these differences are resolved. The problems reflect in large part the absence of a firm underlying architecture for the process models.

We propose that the existing and well defined architectures of enterprise modelling can provide the basis for a suitable architecture. In our proposal, processes are related by their association with established entities in enterprise models - product, project or enterprise entities. An examination of some examples of appropriate indicates that the use of such an architecture can resolve at least some of the existing problems and inconsistencies.

It is concluded that the lack of a sound underlying architecture is the basic cause of the problems and inconsistencies in existing process reference models for the software and system life cycles. The use of enterprise modelling principles to establish suitable architectures is proposed as a resolution to the issue.

References

[1] ISO/IEC 12207: 1995, Information Technology – Life Cycle Processes

[2] SPICE Project (1995), Software Process Assessment – Part 2: A model for process management, Version 1.00.

[3] ISO/IEC TR 15504: 1998, Information Technology – Software Process Assessment - Part 2: A Reference Model for Processes and Process Capability

[4] J.G. Lake (1997), Report on Development of ISO Standard 15288 - System Life Cycle Processes, INCOSE '97, Los Angeles.

[5] ISO/IEC JTC1/SC7 WG7 (1998), Life-Cycle Management — System Life Cycle Processes, Working Draft 3.

[6] IFIP / IFAC Task Force (1997), Generalised Enterprise Reference Architecture and Methodology Version 1.5.

[7] P. Bernus, and L. Nemes, A Framework to Define a Generic Enterprise Reference Architecture and Methodology, Proceedings of the International Conference on Automation, Robotics and Computer Vision (ICARCV'94), Singapore, November 10–12, (1994). also in *Computer Integrated Manufacturing Systems* 9,3 (July 1996) pp 179-191.

[8] ISO 15704: 1998, Requirements for enterprise reference architectures and methodologies.

EFQM/SPICE INTEGRATED MODEL: THE BUSINESS EXCELLENCE ROAD FOR SOFTWARE INTENSIVE ORGANISATIONS

Elixabete Ostolaza and Ana Belen Garcia
Software Process Improvement Guidance Product Line

Summary:

This paper presents the EFQM/SPICE Integrated Model, a framework for business management of software intensive[20] organisations. It offers a descriptive representation of the processes of an organisation that aims to achieve business excellence through continuous improvement. It combines the strengths of two well-known and accepted models: EFQM and SPICE (ISO 15504). The model is the answer to how to apply TQM to software intensive organisations maintaining the EFQM holistic approach, while taking advantage of the effective process-based approach of SPICE for improvement purposes.

The paper focuses on describing: the principles of the model; its structure (based on processes and focused on business results); and the way to apply and integrate it into business operations through tools and techniques such as assessment methodologies and improvement cycles (in particular ESI EFQM/SPICE Assessment Methodology and ESI IMPACT cycle). Additionally, it describes the experience gained with two trials performed with the model.

Software Process Improvement Guidance Product Line.

EUROPEAN SOFTWARE INSTITUTE (ESI). Parque Tecnologico # 204. E-48170 Zamudio, SPAIN. Ph.: ++34-94 420 95 19, fax: ++34-94 420 94 20
Email {gorka, marisa, anabelen, elixabete, zorriketa}@esi.es

[20] Software Intensive Organisation. An organisation which produces software as part of its product development or which develops software for internal operation towards achievement of business performance.

1. Introduction

The Total Quality Management (TQM) concept is gaining momentum throughout all industry sectors as a quality and management philosophy that drives organisations towards business excellence, i.e. towards a sustained improvement of the organisational practices and results that guarantees the continuity of the business. The increasing success of models like those of EFQM and Malcolm Baldrige proves the interest of the international community in TQM.

However, many organisations cannot take full advantage of TQM because they do not have the ability to suit those models to their own particular context (e.g. software companies). On the other hand, improvement of software processes in traditional Software Process Improvement models is not oriented to business goals leading to business excellence.

In order to help overcome these problems, the European Software Institute (ESI) has developed the EFQM/SPICE Integrated Model.

The EFQM/SPICE Integrated Model is a process-based model for continuous improvement that integrates SPI (Software Process Improvement) into the wider context of TQM. For this reason it could be considered as the business excellence road for software intensive organisations. It is the result of merging two well-known and accepted models: SPICE and EFQM.

- The European Foundation for Quality Management (EFQM) and their Model for Business Excellence represents the Total Quality concepts.
- SPICE - Software Process Improvement and Capability dEtermination – (ISO/IEC 15504) is a continuous model for process improvement containing good practices for software engineering.

EFQM is a model for business excellence, which is not specifically oriented to software intensive organisations. It does not provide any help in the difficult task of defining the key processes for a software organisation and how to improve them to achieve concrete business goals.

On the other hand, traditional software process improvement models are not TQM oriented. They manage software processes as isolated from the rest of organisation's key processes. These models focus on measuring the process in order to control it and guarantee its continuous improvement. However, they lose the global picture of the company and often measures are not tied to business goals. The EFQM/SPICE Integrated Model provides a wider and holistic approach by emphasising the business results of process improvement. In this way, quality means meeting customer expectations, not just conformance to a model. In fact, business results will be the starting point for the definition of the business and process goals and improvement opportunities. The link between the software process improvement programme and the business results is therefore guaranteed and the software organisation ensures that improvement is performed in the right way.

The EFQM/SPICE Integrated Model provides software intensive organisations searching for continuous improvement toward business excellence with a unique model, which also takes into account the improvement of software processes. SPICE is a continuous model for process improvement chosen for its flexibility, essential for adapting the processes to specific business goals or company context.

The EFQM approach provides the integrated model with the necessary framework to allow companies to link results with specific business goals and makes a link possible between quality concepts, improvement and business management. In addition, quality systems and business management should not be separate activities but should operate concurrently as part of the overall business system so that they are not inconsistent. Quality is not something that should be added on to a business: the whole business should be managed in a quality way.

The EFQM/SPICE Integrated Model has also inherited from the EFQM Model its important focus on stakeholders: customers – internal and external, current and potential; employees; shareholders; society and subcontractors. These stakeholders are not sufficiently considered by other models or are only considered as part of a contractual relationship, so the organisation tries to be conformant to a set of requirements rather than caring about customer satisfaction. The new model is based on concepts like partnership and establishing win-win relationships with the stakeholders. It recognises the organisation's people as one of its most valuable assets (this is especially true for software intensive organisations). It focuses on customer satisfaction and not just in conforming to customer requirements. This is considered essential for achieving long-term company success.

Alongside the EFQM/SPICE Integrated Model, ESI has developed an assessment method. The main goal of the assessment method is to provide software intensive organisations with a tool, based on the integrated model, to determine their strengths and areas of improvement in their work to continuously improve processes towards business excellence. This is possible thanks to integrated model's approach of processes based on best practices.

2. EFQM/SPICE Integrated Model: Technical Description

The EFQM/SPICE Integrated Model, represented by *figure 1*, is the result of combining the strengths of two well-known and accepted models: SPICE and EFQM.

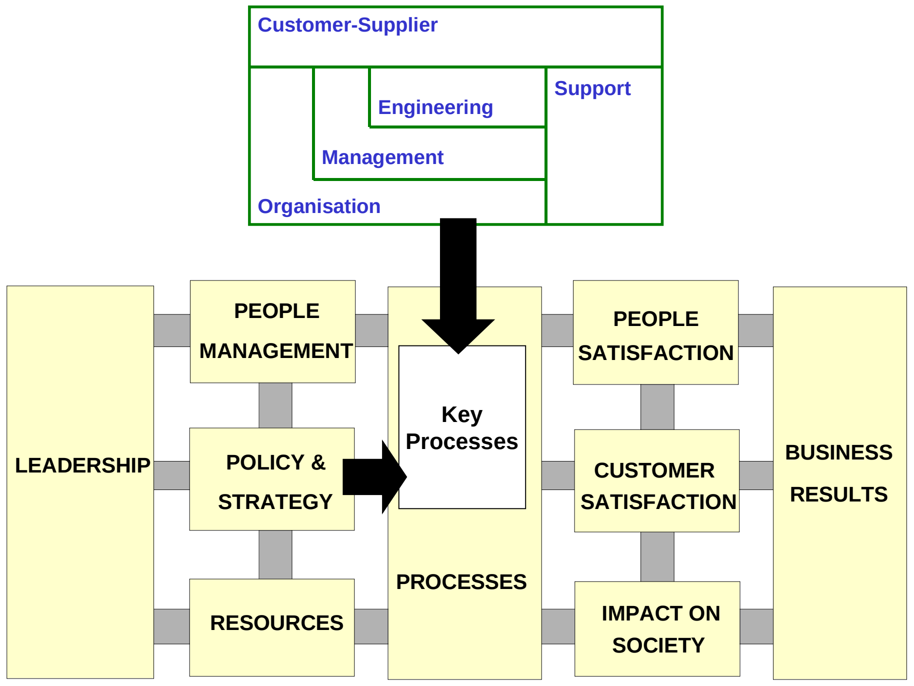


Figure 1 - Structure of the Integrated Model

The integrated model maintains the wider external structure of EFQM but is internally configured like SPICE, based on processes, base practices and work-products. There is a process category for each enabler criteria and in each of these categories there are as many processes as there are subcriteria for the relevant enabler criterion. SPICE processes play an important role in the model as candidates for key processes. A mapping between the candidate SPICE processes and the business goals will determine the key processes for the software organisation.

Like SPICE, the integrated model has two dimensions:

- The processes and results dimension: a descriptive representation of both the processes that a software intensive organisation aiming at TQM should implement and the quantitative data it should gather to check that it is achieving the right results.

- The capability dimension: a mechanism to measure the maturity of the processes, the excellence of the results, and the scope of these maturity and excellence levels.

2.1 Processes and Results Dimension

2.1.1 Processes

There are three types of processes in the integrated model (*figure 2*): enabler processes, key processes and the measurement process. They share a common process structure, which is:

- Process name
- Process purpose
- Process outcomes
- Base practices
- Work products (inputs and outputs).

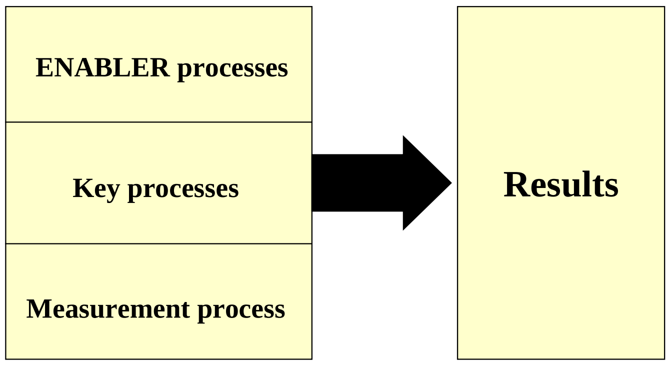


Figure 2 – Structure of the Processes and Results Dimension

Enabler processes

The EFQM/SPICE Integrated Model consists of five process categories, one for each EFQM enabler criteria. These are Leadership criterion process category, Policy and Strategy criterion process category, People Management criterion process category, Resources criterion process category and Process criterion process category. Each process category has as many processes as subcriteria in the corresponding enabler criteria of the EFQM Model.

Key processes

The model provides a set of candidate processes directly extracted from SPICE; processes that characterise software organisations. Processes that are already covered by other enabler processes (corresponding to other EFQM criteria) are not considered key process candidates.
A mapping between the candidate SPICE processes and the business goals, derived from the policy and strategy of the organisation, will determine the key processes.

The SPICE processes proposed by the integrated model as candidates for key processes are: *Supply, Requirements elicitation* and *Operation* processes from the Customer-Supplier category; *Development* and *System and software maintenance* processes from the Engineering process category; *Documentation, Quality assurance*, *Verification, Validation, Joint review* and *Problem resolution* processes from the Support process category; *Project management, Quality management* and *Risk management* processes from the Management process category; and *Reuse* process from the Organisation process category.

Measurement process

The measurement process identifies, collects and analyses data relating to the organisation's results. Its purpose is to demonstrate objectively the performance of the organisation in achieving its business goals and satisfying the stakeholders.

2.1.2 Results

The integrated model contains a set of results, grouped by type and subtypes, to which the measurement process is applied.

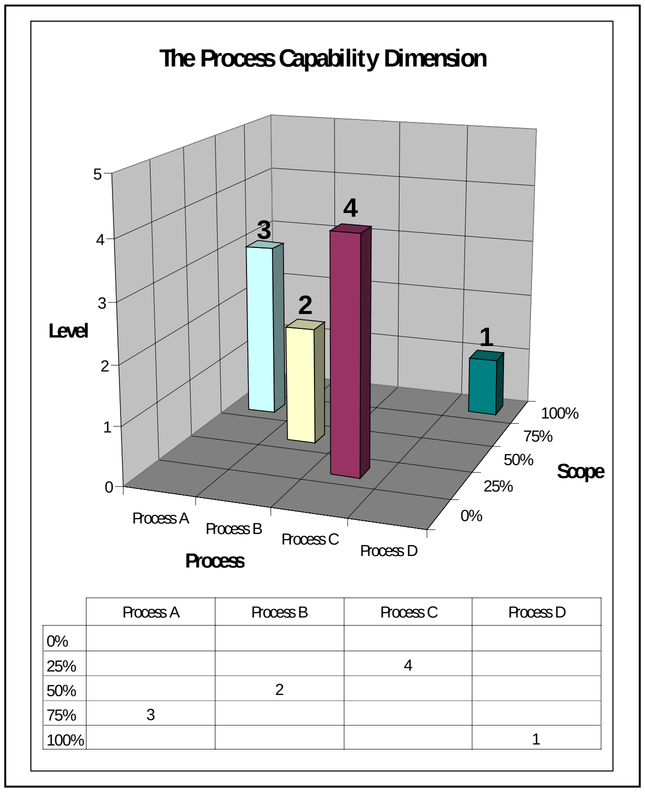
- Each type of result corresponds to an EFQM result criterion.
- Each subtype of result maps with an EFQM result subcriterion.
- Each type of result has two subtypes: the perception subtype, which contains a list of attributes to measure the perception of the results by the customer, people or society; and the performance subtype, with a list of attributes that measure the performance of the organisation in satisfying the needs of customers, people, society and shareholders.

2.2 Capability Dimension

The EFQM/SPICE Integrated Model’s second dimension, the capability dimension, has two aspects: one for processes and another for results.

The **‘process capability dimension’** is defined on the basis of the combination of two factors (see *figure 3*):

- LEVEL – measures the degree of excellence of the process performance and implementation approach. Like SPICE, there is a six point ordinal scale that enables capability to be assessed from the bottom of the scale, Incomplete, through to the top of the scale, Optimising.
- SCOPE – measures the degree of deployment of the process throughout the organisation. Like EFQM, there is a percentage scale that enables the scope of the application to be assessed from ‘not applied in relevant areas’ to ‘applied in all relevant areas and activities’.



	Process A	Process B	Process C	Process D
0%				
25%			4	
50%		2		
75%	3			
100%				1

Figure 3 – The Process Capability Dimension

The **'results excellence dimension'** is also defined on the basis of the combination of two factors:

- EXCELLENCE - measures the excellence of the results based upon a set of five attributes: *Pertinence, Trends, Targets, Comparisons and Cause*. Like EFQM, there is a one hundred point cardinal scale that enables excellence to be assessed from the bottom of the scale (0), *No Results*, through to the top of the scale (100), *Best in Class.*
- SCOPE – measures the degree of coverage of the organisation's relevant missions, business goals, areas and activities. There is a percentage scale that enables scope of the results to be assessed from '*not applied in relevant areas*' to '*applied in all relevant areas*'.

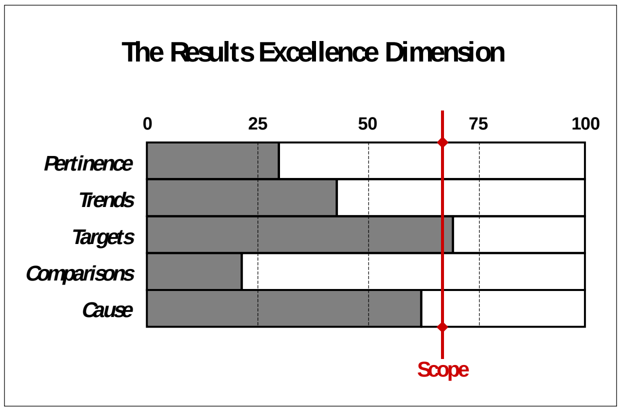


Figure 4 – The Results Excellence Dimension

3. Applying the EFQM/SPICE Integrated Model

The EFQM/SPICE Integrated Model provides a framework for business management of software intensive organisations. It offers a descriptive representation of the processes of an organisation that aims to achieve business excellence through continuous improvement.

The model itself is a static reference point, a structured group of good practices for organisation and process management, that allows top management to understand what TQM means for their software company. Applying this model to an organisation implies combining it with a set of dynamic tools and techniques that integrate TQM and SPI into business management.

The EFQM/SPICE Integrated Model could be used solely for assessment purposes or for business planning and improvement. The latter incorporates assessment, and particularly self-assessment, as a strategic step in business management based on TQM concepts. To support the application of the model, ESI has developed:

- An Assessment Methodology (assessment method description, plus training material and tool-set to support the assessment execution).

- An adaptation of the ESI Business Improvement Cycle, IMPACT (Initiate, Measure, Plan, Act, Confirm, Transfer), to an organisation aiming to implement TQM.

A description of the integration of the Model with the IMPACT continuous improvement lifecycle is included below, in order to clarify the application of the EFQM/SPICE Integrated Model in the business management context.

- INITIATE business improvement. The aim of this phase is to guarantee that the business improvement programme is closely linked to the organisation's business needs and objectives. From the EFQM/SPICE point of view, this implies a review of the organisation from the stakeholder perspective (customers, employees, society, shareholders, and partners), in order to clarify and define the organisation's mission and objectives in relation to its stakeholders. The business objectives will be quantified in terms of the RESULTS of the EFQM/SPICE Integrated Model, so that the improvement cycle is driven by the optimisation of these business results. In addition, the key processes for the business will be selected as a result of mapping the SPICE candidate processes of the integrated model with the business goals. Finally, this phase of clarification and definition of the business mission and improvement goals should be performed jointly with the business planning activities. This will ensure that quality and improvement are understood as an embedded part of the global strategic business planning rather than as an external constraint that is later added to the system.

- MEASURE current situation. The aim of this phase is to understand the current situation of the organisation, in relation to the previously defined business goals and target results. This implies performing an assessment using the EFQM/SPICE Integrated Model as the framework against which the organisation is compared. Following the TQM philosophy, the assessment should be carried out in two steps:

    - Assessment of the RESULTS: what the organisation is currently achieving in relation to the quantifiable results that characterise the business objectives and that were defined in the INITIATE phase. The performance gap is determined between the targets defined in the INITIATE phase and the current results.

    - Assessment of the PROCESSES: based on the principle that the organisation's processes generate the organisation's results, this assessment determines the cause of the performance gaps found in the results assessment. Both key processes and enabler processes are assessed. To reduce the scope and cost of the assessment, a preliminary study can be performed to determine the set of processes that are most likely to affect the business results and business goals, so that only those processes are assessed. ESI has developed an interdependency analysis between the criteria and sub-criteria of the EFQM Model to support the selection of adequate processes to assess.

  The result of the assessment is a description of the strengths and areas for improvement of both the results and the processes relating to the organisation's business goals.

- PLAN the improvement. The aim of this phase is to prioritise and plan the actions to be taken, based on the assessment results and business goals. Actions will mainly include improvement of existing competencies and processes or creation of new processes. The EFQM/SPICE Integrated Model could be used as a source of good practices for defining the process improvements. A Measurement Plan should be developed following the practices recommended by the Measurement Process of the EFQM/SPICE Integrated Model. This will be essential for the future verification of the improvements and for evaluating the overall effectiveness of the business management system in fulfilling the business goals.

- ACT to improve. The aim of this phase is to implement the actions following the defined plan. Managing the change is a critical step in making improvements happen. It is important to maintain continuous control over the implementation activities in order to avoid deviations. This phase is not covered within the scope of the EFQM/SPICE Integrated Model.

- CONFIRM and sustain the improvement. The aim of this phase is to review and verify improvement and to ensure that it will be sustained in the future. The verification method will be the re-assessment of the processes and results of the EFQM/SPICE Integrated Model that have been involved in the improvement. The assessment of the quantifiable results will be possible thanks to the application (during the ACT phase) of the Measurement Plan that was designed during the PLAN phase. This plan will have provided the appropriate data to evaluate fulfilment of the objectives of the plan. The activities and controls necessary to ensure that the improvements will be sustained in the future should be analysed and put into action.

- TRANSFER the benefits. The aim of this phase is to complete the improvement life cycle by documenting the benefits obtained, the experiences gained and the lessons for the future. This makes it possible to establish and maintain the organisational knowledge, culture and assets. These are the key elements to long-term successful improvement. The TRANSFER phase of IMPACT implies achieving these goals not only in an individual organisational context but also in a collaborative international context, sharing experiences for the benefit of all. This is an important principle of the EFQM interpretation of TQM. ESI provides a Repository of Experiences that allows the institutionalisation of improvements across Europe.

In summary, the EFQM/SPICE Integrated Model provides a process based framework for software intensive organisations to implement a management strategy for software process improvement, aimed at business excellence, based on continuous improvement cycles.

3. Validating the EFQM/SPICE Integrated Model

The European Software Institute (ESI) has carried out in the second semester of 1.998 two trials of the integrated EFQM/SPICE Model in the Software Departments of two large Organisations.

The ESI objectives of the trials were to validate the efficiency of the EFQM/SPICE integrated model in a real context and to evaluate the EFQM/SPICE assessment method.

The Software Departments objectives were to obtain a picture of the actual situation of their departments. As a trial result they got a report of three parts:

- A list of the main strengths and areas for improvement.

- For each of the assessed process, a complete list of the strengths and areas for improvement found and, the profiles of each process.

- A list of recommendations for starting an initiative of improvement.

The first step was to ask Department Managers to describe their business goals. Once the business goals were clearly identified, the assessor team first selected from the candidate SPICE key processes the appropriate ones to be assessed considering the defined business goals and next, the appropriate Enabler processes to be assessed were selected. The process selection method applied for the SPICE processes was the one that ESI has developed to define their BIG-guides[21]. The Enabler processes were selected taken into account the software department context (is the organisation involved in a total quality management initiative? Does the software department have an independent budget? Do they have autonomy to define the human resource management policy?,…).

The goal of the assessment was to get a picture of the current situation of the Software Departments, both from the organisational point of view - focusing on leadership, people management and financial aspects in an EFQM assessment manner – and from the projects point of view – assessing design, project management, relationship with customers for the project,… in a SPICE like manner -. That is, the goal was to perform an EFQM and SPICE integrated assessment.

Positive aspects observed when performing an integrated EFQM/SPICE assessment were:

- The ability to refine or redefine the initially stated business goals with the assessment results.
- The assessments highlighted the extent of the role which software has in an organisation.
- The assessments enabled organisational managers to get a better understanding of how software is developed.

[21] BIG-guides: Series of Business Improvement Guides that relate the software process improvement (based on SPICE) to static business goals. ESI has developed three BIG guides: BIG-TTM – with Time To Market as business goal -, BIG-ISO – with ISO 9000 as business goal - and BIG-CMM – with CMM level 2 as business goal -.

- The assessment demonstrated gaps between organisational business management and the software management raising awareness of the need to manage software based on organisational needs.
- The SPICE like process assessment approach made easier to software practitioners to interpret the EFQM criteria.
- The SPICE like process structure of the EFQM/SPICE integrated model provides an interpretation of EFQM for software intensive organisations that is very useful to provide strengths and improvement areas for the software organisation or department assessed.

Some weaknesses detected during the integrated assessments:

- Some organisational processes related with organisational issues such as leadership or policy and strategy are difficult to assess at the department level when the full organisation is not involved in a total quality management improvement initiative. The organisation top management commitment and participation is therefore essential for the complete success of this type of initiatives.
- The capability dimension of ENABLER processes is, in some cases, complicated for rating purposes. Further work on this aspect is required to improve the model.

4. Conclusions

The EFQM/SPICE Integrated Model presented in this paper provides software intensive organisations with a unique model that combines the strengths of two well- known and accepted models: EFQM and SPICE. The model can be used for assessment purposes or for business improvement and planning purposes. Its process approach, taken from SPICE, makes the model a flexible tool.

4.1 Added Value with Respect to SPICE

The added value of the EFQM/SPICE Integrated Model with respect to SPICE is:

- It offers a holistic process description of the software organisation, so it covers organisational issues that are partly or completely outside the scope of SPICE. For example, this includes leadership, people management and financial aspects.

- Software process improvement and quality activities do not follow a separate direction to business management activities. They should be integrated as part of the global management strategy so that quality is built into the system rather than added to it as an external requirement.

- The EFQM/SPICE Integrated Model focuses on business results and implies measuring those results. SPICE only looks at measuring processes to control them and guarantee their continuous improvement at process level. It does not fully address the global organisational picture. The new model measures business results as the starting point for defining the appropriate measures to control and improve the processes. Measuring the results provides valuable information about the real business improvement.

- The new model shows a clear relationship between how the software intensive organisation operates (processes) and what it achieves (results). It allows an improvement approach clearly driven by the business needs and goals. SPICE is not able to guarantee so clearly that the right improvements are being implemented.

4.2 Added Value with Respect to EFQM

The added value of the EFQM/SPICE Integrated Model with respect to EFQM is:

- The EFQM/SPICE Integrated Model is an interpretation and adaptation of the general EFQM Model for the software development world. It facilitates its applicability to these kind of organisations by making it easier for software practitioners to understand.

- It shows what TQM means for a software intensive organisation. It describes a set of good practices for implementing the EFQM Model in a software intensive organisation.

- It is process focused, so it is possible to apply it to software intensive organisations (in assessments, improvement programmes, etc.) in a more flexible way (e.g. considering only the processes, which have an impact on a sub-set of results).

- The new model includes a set of clearly defined capability levels for the processes that allows a more objective assessment of the maturity of the processes (sub-criteria in the EFQM Model). In addition, the capability levels define a way to manage and improve a process and so support the definition and implementation of improvement programmes.

5. Bibliography

Conti, Tito Building Total Quality. A guide for management. English language edition 1993. ISBN 0 412 49780 8

Conti, Tito Organizational self-assessment. First edition 1997. ISBN 0 412 78880 2

EFQM European Foundation for Quality Management. Self-Assessment 1997. Guidelines for Companies

SPICE ISO/IEC TR 15504 Software Process Assessment's Parts 1-9, Technical Report type 2, 1998

Rementeria, Santiago Software management effectiveness in a wide organizative context. Paper, 1997

Garro, Inigo Improvement for the business. Iñigo Garro & Giuseppe Magnani. European Software Institute (ESI). Paper, 1998

A Reading-based Scenario for

ChaRacterising and Exploiting Process components

Maria Tortorella* and Giuseppe Visaggio**

* Faculty of Engineering, University of Sannio

Palazzo Bosco Lucarelli, Piazza Roma, 82100 Benevento, Italy

(martor@ingbn.unisa.it)

** DIB - Dept. of Informatica, University of Bari,

Via Orabona 4, 70126 Bari, Italy

visaggio@di.uniba.it

Abstract

The improvement of the existing software processes requires the adoption of innovative process components, and the exploitation of the process components that are already used in other contexts. But it can happens that only an informal description of the process components is available. Therefore, before adopting a process component, it is necessary to formalize it for suitably comprehending the information available and evaluate if they can be candidate for integration into a software process to be improved. The authors of the paper propose a characterization framework for helping to comprehend and candidate a process component. This paper describes an experimentation to validate the framework by using a scenario. The unexpected result obtained indicated the framework more effective without using the scenario. This event encouraged the authors to

make a more probed analysis of the available data. Two lessons learned resulted by this analysis: the first one assured that the use of the framework is as much effective as more difficult the source of information to be understood is; the second one suggested some modifications to the framework to make the scenario to be more effective.

1. Introduction

The requirements of improvement of the software product quality imply the continuous improvement of the software processes [1], [2], [9] and [14]. This can be achieved by integrating innovative process components into the working software processes to be improved. The expression *process component* is used to indicate either a guideline (a simple set of rules), or a technique (an algorithm or a series of steps whose execution requires some knowledge and competence and produces a certain effect), or a method (a particular management procedure to apply techniques), or a simpler process (a set of methods and interrelationship to combine each other and necessary to reach a specific goal).

The improvement intervention can be referred to the following aspects: the extension of an existing process by introducing one or more innovative process components; the substitution of one or more components of the process with innovative ones that are semantically equivalent; the modification of one or more components in the software process. This definition includes also the aspects of construction of a new software process.

There exist a fair amount of process components available for improving a software process. However, it is not always apparent to the manager which of those to invest in, and whether they pursue the improvement goals established and are integrable into the working process to be improved. The improvement intervention involves the following tasks:

1. identification and description of the improvement goals and the characteristics of the process components in order to be integrated into the process to be improved and to be adequate to the production environment;

2. identification of the process components that are to be substituted or added in order to reach the predefined improvement goals;

3. definition and/or identification of the process components to be integrated;

4. evaluation by using a quantitative approach of the quality and integrability of the process components identified and risks their adoption involves.

Tasks 1 have already been faced in the literature [1], [2], [14]. Task 2 is committed to the manager decisions. Task 3 involves the searching in literature or in other information sources. Task 4 is treated in this paper.

The exploitation of the process components requires they are understood in order to be evaluated and candidate, choose the most suitable one between the ones candidate and integrate it into the software process. This requires the extraction of complete and correct information about process components from the documentation. But, unfortunately, This task is not always easy owing to the lack and ambiguity of the information published. Therefore, it is difficult to completely comprehend them and all the attributes necessary for evaluating their conformity and adequacy to the other process components to be interfaced.

These problems are still scarcely faced in the existing literature. In [10], [11] and [12], Song and Osterweil propose an approach for comparing project and development methods but this depends strongly on the process component being evaluated and takes no account of whether it could be integrated and made operative in the process to be improved. Discussion of these problems in greater depth is made in [13], while the lack of immediate applicability of the above approach has been demonstrated in [5], the authors were only able to apply the evaluation schema to High Performance Systems after substantial modifications had been made.

The approach REP (chaRacterising and Exploiting Process component) aims to extracting and exploiting innovative process components in order to construct a new process or improving an existing one [6]. REP is based on a *characterization framework* useful for the analysis and evaluation of the process components. The framework can be modified and specialized to the particular context the process component has to be used in, [6]. In fact, the improvement requirement of a process requires information that depends on the general goals of a process and the specific ones of a project. Then, the framework of REP can be modified on the basis of the specific goals of the software process and project. Besides, it guides to comprehending the analysed technology and helps to identify the inconsistencies, ambiguities and incompleteness of the documentation available.

The paper presents a controlled experiment aiming to verify whether the characterisation framework improves comprehension of the process component being analysed and whether the evaluation can be further on improved by using an approach based on reading through a scenario. The results obtained showed that the reading from an informal description to construct an applicable formalism presents problems of correctness and completeness that are not easily solvable by using a technique based on scenario. The paper presents some shrewdness to be adopted when a scenario is defined.

The following section briefly describes the REP approach and its *characterisation framework*. Section 3 presents the experimental design and the probed analysis. Finally, the last section states lessons learned and further work.

2. The REP approach

The REP approach aims to support a user to innovate a software process by adopting a process component described in literature [13]. It is based on the *formalization* and *characterization* of the process components to be analysed.

The formalization task consists on analysing the documentation available about a process component, interpreting the ambiguous and incomplete information, and describing the process component in a formal and unambiguous model. This task is pursued by adopting a Modelling Formalism chosen by the software engineer. The one adopted in the REP approach is the one supported by the Process Conceptual Modeler (PCM), a tool in the PROMETHEUS (PROcess Model Evolution Through Experience Unfolded Systematically) environment [4].

The characterisation task involves the instantiation of a *characterisation framework* whose aim is to help a software engineer to recognise those process components that satisfy the requirements of the improvement process, evaluate them and choose the one best suited to his purposes from among a set of process components with similar features [13]. Then, it can act as a guide to formalising process components and helps to identifying inconsistency, ambiguity and incompleteness in the documentation published in literature

The *characterisation framework* synthesises into homogeneous categories all the information necessary for evaluating the integration of a process component into a software process to be improved. The aim is to verify if the characteristics of the process component analysed are conform to the correspondent ones in the process to be improved. The main categories considered are the following:

Input, to define the characteristics of the objects necessary for the process component to operate. Seven questions are considered to investigate about the typology of the input deliverables necessary to the process component to operate and their mappability, conformance and adequacy to the deliverables available in the process to be improved;

Output, to define the characteristics of the objects the process component produces. Even in this case, seven questions are considered to analyse the conformity of the deliverables produced by a process component as output, to the deliverables the process to be improved expects from it;

Emphasis, one question has to be answered to identify the process component goals and their conformance to the goals of the process improvement;

Domain, one question is considered to collect the characteristics that are to be satisfied by the input objects in order for processing by the process component to be possible;

Environment, two questions are used to analyse the starting state of the external environment necessary for the process component to operate, and the ending state produced by its application;

Techniques, one question is considered to outline the technological characteristics of the process component;

Formalisation, one question has to be answered to define the rigorousness with which the process component is expressed;

Quality, six questions are used to express the quality of the process component and output objects produced: the automation level of the process component and its possible improvement, if quality parameters and econometric models are used, and its scientific and experimental maturity.

For brevity, the full characterisation framework is not shown here. A detailed description can be found in [6].

The framework can be considered as composed of a general part that can be reused in any context and a specialised one that has to be adapted to the particular kind of process component to be analysed. The general part is composed of questions general purpose that can be applied to every context and any kind of software process. The specialized part is composed of questions that mould the framework to the particular context of the software process to be improved. For example, the experimentation illustrated below was an application to the field of the reverse engineering and the framework included questions specialized to this context, i.e. questions investigating the possibility to recover the unstable knowledge residing in the minds of men who have a good working knowledge of the programs, the possible modification of the input deliverable, the scientific and experimental maturity, and so on. Being composed of a general part and a specialized one, the framework can evolve, be improved as new experience is gained, and modified to consider new aspects as occasion may arise. From this point of view, the *characterisation framework* can be considered as an *accumulator of experience* and it is reusable, as a framework should be, and can be specialised for each application domain using the experience accumulated concerning the target domain.

2.1. The scenario

In order to support an evaluator to correctly apply the REP approach, an instrument based on reading was defined. This instrument aims to guide the evaluator to extract the needed information from the documentation of a process component and minimize the subjectivity when answering the questions of the framework. Then, a *scenario* of REP was elaborated. It indicates to the evaluator exactly which information are needed for evaluating a process component. In it, the construction of the Process Model occurs together with the instantiation of the characterization framework. In fact, the scenario is based on the assumption that in order to answer a question of the framework, it is not necessary to analyse all the objects in the process model. Then, each question is preceded just by the formalization steps defining the process model objects necessary to answer it.

Table 1 shows a fragment of the scenario to answering the first question of the framework. Question 1 is referred to the input mappability whose evaluation regards the information about the deliverables input to the analysed process component and their structures and it is preceded form Steps 1 and 2 aiming to formalize this information.

Table 1 – Scenario fragment for the input category

Step 1. Identify the components for each Input Deliverable and describe them in the DBS

Step 2. Create the Deliverable Report of each Input Deliverable in the DBS. It will contain the following:

– the deliverable structure: the deliverables that are simple are to be managed like information tokens; the compound deliverables are to be described by using the chain, alternative e repetitive operators (Deliverable Structure);

– a textual description of the deliverables (Deliverable Description).

I1. Are the input deliverable required by the process component mappable to those available?

M1. Input_mappability

Domain of values : {not mappable; partialy mappable; fully mappable; under-mappable; over-mappable }

Scenario: Identify at the context level the input deliverable DI_P:

– Substitute the deliverable in DI_P that are at an abstraction level higher than the available deliverables in DI_A, with their sub-components described in the DBS

– Substitute the deliverable available in DI_A that are at an abstraction level higher than the input deliverables in DI_P, with their sub-components

– Verify the equivalence of the deliverables in DI_P to the ones in DI_A:

$DI_P \cap Di_A \neq \emptyset$:	Input_mappability = partialy mappable;
$DI_P \cap Di_A = \emptyset$:	Input_mappability = not mappable;
$DI_P = DI_A$:	Input_mappability = fully mappable;
$DI_P \subset DI_A$:	Input_mappability = over-mappable;
$DI_A \subset DI_P$:	Input_mappability = under-mappable;

3. Experimentation

The experimental subjects involved in the experimentation were students attending Software Engineering II, a course in the fifth year at the Department of "Informatica" of the University of Bari.

The analysis of the experimental results was made from two perspectives. The first perspective aimed to verify if the characterization framework could be considered an adequate tool for better comprehending the informal description of a process component. The second perspective aimed to evaluate the effectiveness of the scenario as a reading technique helping to make uniform the comprehension of the different analyzers. The analysis from the first perspective is briefly described below. A detailed description is given in [6]. This paper discusses in details the analysis made from the second perspective.

3.1. Experimental design

All the experimentation considered six independent variables.

1. The *training* to the use of the characterization framework. It took place through theoretical lessons and practical sessions. Table 2 lists the topics treated during an initial training, and the hours were dedicated to each topic. The initial training was followed by an initial experiment during which the experimental subjects could apply the knowledge gained to a simple process component. The comparison experiment occurred after further training, necessary to solve the difficulties met in the initial experiment. Two more complex process components were analysed during the comparison experiment. The training was the *treatment variable* for the first perspective of the experimentation.

Table 2 – Topics discussed before the initial experiment

Topic	**Hours**	**Groups Attending**
Process modelling	4	A, B, C, D
Basic concepts regarding the need for, and meaning of, the process innovation	1	A, B, C, D
Methods for reusing process components published in literature and developed in research environment	3	A, B, C, D
The *framework*, its contents and use in evaluating a process component	2	A, B, C, D
The *scenario* based on reading techniques	8	C, D

2. The *team composition*. The experimental subjects were randomly grouped into four groups, A, B, C and D, of nine people each. Groups A and B were asked to apply the REP approach by using a Checklist. Groups C and D applied the scenario. The four groups were trained differently. The third column in Table 2 shows which groups was trained in each topic. This variable analysed the *selection effect* and was used to show if, with regard to the particular method applied, the results obtained were influenced by the composition of the teams.

3. The *analysis round*. Three rounds taking place in three different days. The initial experiment was performed in the first round, TRIAL; while the comparison experiment was carried out in two rounds, ROUND1 and ROUND2. This variable measured the *maturation effect*, and was considered to analyse if, after an adequate training to the use of the framework, all the following evaluations of process components provided the same results.

4. The *process components to be analysed*. Three process components were analysed: LWD [7] was analysed during the TRIAL round by Teams A and B, CLV [3] and SND [8] were considered during ROUND1 and ROUND2. This variable was considered to measure the *instrumentation effect* and it aimed to verify if, after an adequate training, the evaluation of the process components was influenced by its complexity and application domain.

   It is important to highlight that the two process components, CLV and SND, had characteristics that were remarkably different. In fact, CLV was realized in an academic context and considered mainly scientific aspects, while SND was defined in an industrial context and paid more attention to technical aspects. The same opinion was expressed by the experimental subjects, which highlighted, also, that CLV was more difficult to be understood than SND**.**

5. The *component order*. As it is shown in Table 3, the four teams did not analyse the two process components of the comparison experiment in the same order. This variable needed to analyse the *presentation effect* and if the order in which the different process components were evaluated altered the results.

6. The *approach adopted*. Two approaches were adopted, the Checklist and the Scenario. This variable was analysed to compare the results obtained when the scenario was applied and when it was not. This variable was the *treatment variable* for the second perspective of the experimentation.

Table 3 – Experimental design

Round	Process component	Checklist teams	Scenario teams
ROUND1	CLV	A	C
	SND	B	D
ROUND2	CLV	B	D
	SND	A	C

Three dependent variables were considered *correctness*, *completeness* and *similitude rate*.

Correctness indicates the rate of the correct answers (*#correct answers*) given by the experimental subjects on the total number of questions answered (*#answered questions*). While *completeness* measures the rate of the answers given by the experimental subjects (*#answered questions*) on the total number of questions in the framework (*#questions*), in this case 26:

$$correctness = \frac{\#correct\ answers}{\#\ answered\ questions} \qquad completeness = \frac{\#answered\ questions}{\#questions}$$

The *similitude rate* evaluates how homogeneously the experimental subjects answered to the question of the framework. Let S_1, S_2, …, S_k be the k experimental subjects, and C_1, C_2, …, C_{26} be the 26 questions of the characterization framework. The similitude rate, ρ_p, of the generic question C_p is given by:

$$\rho_p = \left(\sum_{i=1..k-1;\, j=1+1..k} \rho_{i,j}\left(S_i, S_j\right) \right) / \frac{\kappa!}{2!(k-2)!}$$

where:

- ρ_{ij} *(S_i, S_j)* = 1, if S_i and S_j give the same answer the question $C_{P;}$
- ρ_{ij} *(S_i, S_j)* = 0, otherwise.

The values of the similitude rate can vary between 0 and 1. If k=0 then ρ_p=1 and if k=1 then ρ_p=0. Moreover, the highest value for question C_P is obtained when all the experimental subjects give the same answer to C_P, while the lowest value is obtained when a couple of subjects giving the same answer does not exist.

3.2. First Perspective Analysis

Figure 1 depicts the correctness and completeness rates obtained for the three process components analysed through box-plots. The results obtained for LWD in the initial experimentation are worse than the ones obtained for CLV and SND in the comparison experiment. This proves that a better knowledge of the framework improves both the correctness and completeness values.

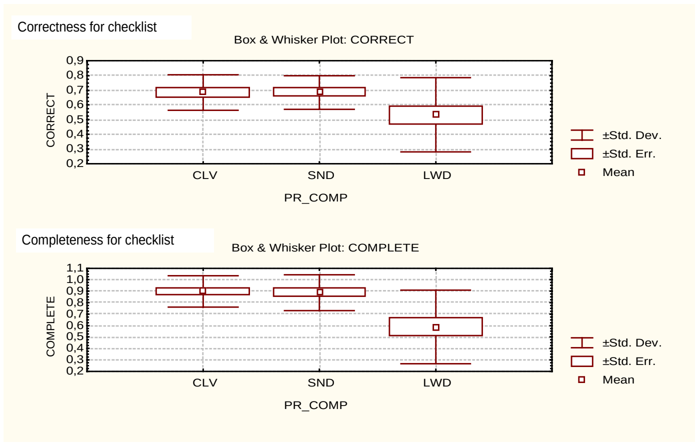


Figure 1 – Correctness and completeness rate for the checklist approach

The statistical analysis was performed by one-way analysis of variance (ANOVA analysis) for all the independent variables, to identify the individual variables that could explain a significant amount of variation in the rates. The ANOVA analysis evaluates if one or an aggregate of independent variables has a significant effect on a dependent variable through the calculation of the P-value. When P-value, evaluated for one or more independent variables and a dependent variable, is less that 0.05, it means that the independent variables have a meaningful effect on the dependent variable.

Table 4 highlights that for both the correctness and completeness, the only independent variable that evaluated a P-value lower than 0.05 is the treatment variable, the *training*. This means that the training has a significant effect on the completeness and correctness values obtained for LWD in the TRIAL experiment and those obtained for CLV and SND in the comparison experiment, after further training. Then, the improvement obtained was influenced by the experimental subjects' better knowledge of the framework. On the contrary, P-values higher than 0.05 were obtained for the other four independent variables. This means that any improvement obtained in the evaluation of the two process components was not influenced by the experimental subjects' greater experience in understanding process components (*maturation effect*), by the characteristics of the process components analysed (*instrumentation effect*), by the team composition (*selection effect*), and the order in which the process components were analysed (*presentation effect*).

Then, when sufficient knowledge of the framework was gained, and if it did not change, any independent variable considered did not affect the validity of the experimentation.

Table 4 - Correctness and completeness P-values of the old experimentation

1.15.1.3 Independent Variable	1.15.1.4 Components compared	***Correctness P-value***	***Completeness P-value***
Training (treatment variable)	1.15.1.4.1 LWD - SND	0.0025	0.0076
	LWD - CLV	0.0032	0.0051
Analysis round (maturation effect)	Round1 - Round2	0.1319	0.8547
Process components (instrumentation effect)	SND - LNV	0.7915	0.8547
team composition (selection effect)	Group A - Group B	0.6399	0.0512
Component order (presentation effect)	CLV Group A – CLV Group B	0.46	0.2144
	SND Group A – SND Group B	0.188	0.1653

3.3. Analysis of the results with scenario

Figures 2a and 2b depict the correctness and completeness rates obtained in the experimentation for the approach using the scenario. In this case, the correctness and completeness rates are lower than those obtained for the checklist. The justification of the completeness results comes is that a step of the scenario could be performed only if all the previous steps were applied. This process requires

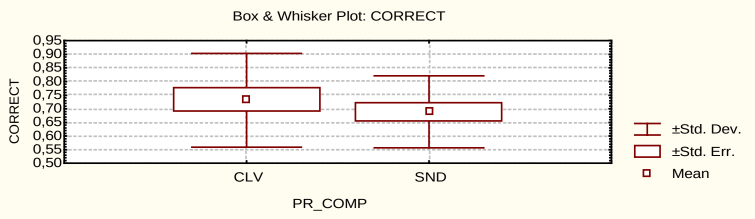


Figure 2a - Correctness for approach based on Scenario

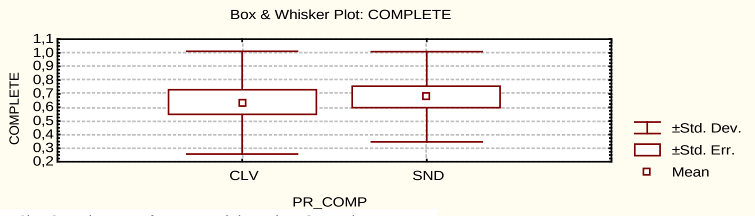


Figure 2b - Completeness for approach based on Scenario

Figure 2 –Correctness and completeness rate for the scenario approach

more than four hours to be completely applied. Therefore, the worst completeness values were probably due to the greater tiredness of the experimental subjects using the scenario.

Table 5 shows the ANOVA analysis for the correctness and completeness for the second perspective of the experimentation. None of the independent variables had effect on the correctness value (all the P-values are major than 0.5). While, the variable *analysis round*, checking the *maturation effect*, influenced the completeness values (P-value<0.05). This meant that an improvement on the completeness value was influenced by the experimental subjects' greater experience in applying the scenario, and the major experience required less time for applying it during ROUND2. Moreover, the completeness value was influenced by the variable *component order*, but, only when the most difficult process component, CLV, was analysed. This confirms that the major benefits of the scenario can be obtained when the process component to be analysed is very difficult, and this result is reached with the major experience. Table 5 shows that the sixth independent variable, *approach applied*, has a significant effect on the completeness, while do not affect the correctness. The effectiveness on the completeness value was obtained because the approach based on the scenario required more time to be applied than the one based on the checklist.

Table 5 - Correctness and completeness P-values for the scenario approach

1.15.1.5 Independent Variable	1.15.1.6 Components compared	***Correctness P-value***	***Completeness P-value***
analysis round (maturation effect)	Round1 – Round2	0.9635	0.023
process components (instrumentation effect)	SND – CLV	0.6979	0.8054
team composition (selection effect)	Group C – Group D	0.132	0.1165
Component order (presentation effect)	CLV Group C – CLV Group D	0.3542	0.0009
	SND Group C – SND Group D	0.2412	0.6902
1.15.1.7approach applied	**Scenario-Checklist**	0.3762	0.0052

Figure 3 shows the values of the similitude rate obtained for CLV and SND and both the approaches. The worse values of the similitude rate were obtained with the scenario, and the answers given by the experimental subjects applying the checklist had an higher similitude rate than those applying the scenario.

The better results obtained for the approach applying the checklist were unexpected and, as this was not explicable by a superficial analysis, the authors were induced to probe into the analysis of the available data.

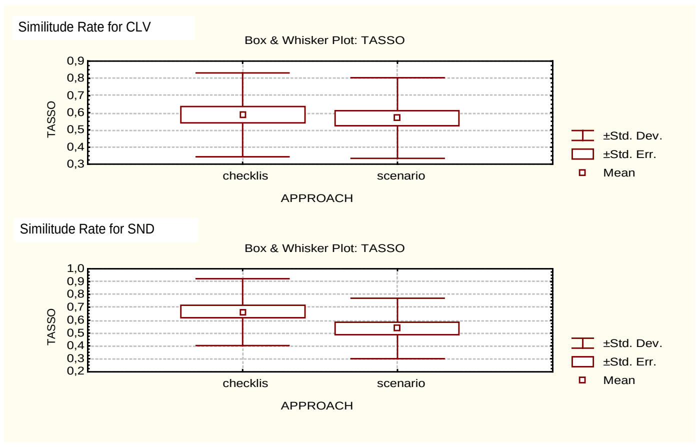


Figure 3 – Similitude Rate of the answers provided for both CLV and SND

3.4. Probed analysis

The poor results obtained for the approach based on the scenario were due to the poor usability of the scenario in the context where it was applied during the experimentation and the time the experimental subjects had at their disposal. Figure 4 shows that the time spent by the experimental subjects applying the scenario was longer than the time spent by the other ones. The figure shows that

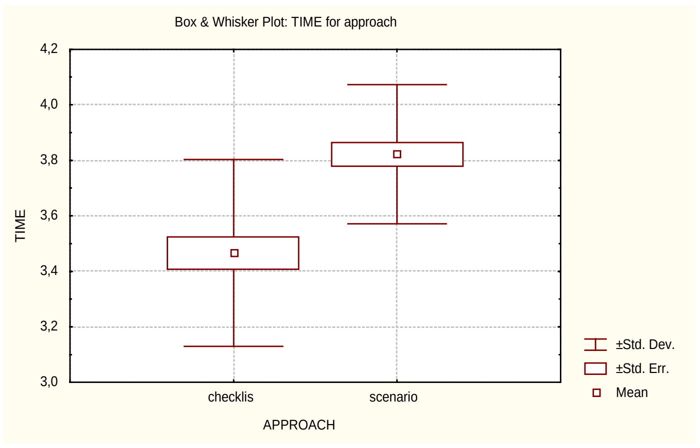


Figure 4 – Analysis of the time spent for the approach adopted

the medium time used to apply the checklist was of about 3.47 hours, and a good value of completeness was reached (medium value equal to 0.89 about), while the one to apply the scenario was about 3.83 hours without reaching a good completeness value (medium value equal to 0.65 about).

Moreover, Table 6 shows the ANOVA analysis of the variable *Time spent*. It highlights that the approach applied had a significant effect on the dependent variable *Time spent* (P-value < 0.05). The time spent to apply the Scenario increased with the complexity of the process component. In fact, Figure 5 highlights that CLV required more time when the scenario was applied than SND, and, as stated above, CLV was the most difficult process component to be analysed. On the contrary, the analysis of CLV and SND using the checklist required about the same time. Even the ANOVA analysis in Table 6 shows that the approach applied had a significant effect on the time spent when CLV was analysed by using the scenario (P-value < 0.05). The increasing time necessary

Table 6 – ANOVA analysis for the dependent variable Time spent

1.15.1.9 Independent Variable	1.15.1.10 Components compared	***Time P-value***
Approach applied	Scenario – Checklist	0.000009
	Scenario – Checklist (SND)	0.0136
	Scenario – Checklist (CLV)	0.0002
Process component to analyse	CLV-SND	0.3277
	CLV-SND (Checklist)	0.7787
	CLV-SND (Scenario)	0.0116

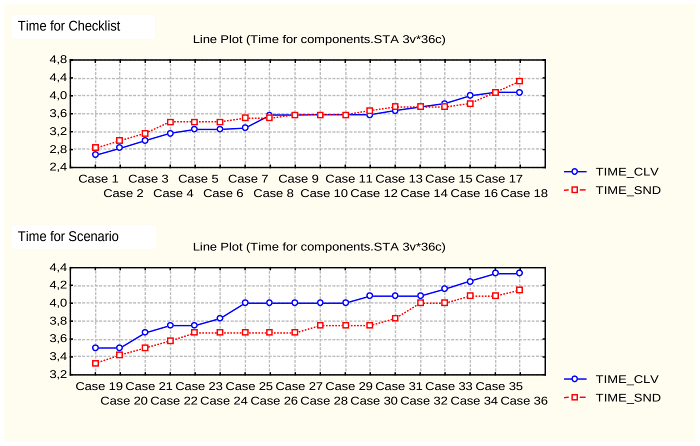


Figure 5 – Analysis of the time spent for analysed process component

when the process component is more difficult can be caused by the usability of the scenario and, sometime, the difficulties to provide univocal and well-defined answers some questions of the framework.

A more careful analysis of the documentation about CLV and SND revealed that some questions of the framework could not be answered because the information published and useful to answer them were either ambiguous or incomplete. The experimental subjects that applied the checklist were not constrained to choose their answers in a predefined value domains, and they could give answers like *ambiguous* or *not defined* when a clear answer was not available. In particular, the questions that could not be precisely and correctly answered by using the scenario were eight for SND and nine for CLV. The major complexity of CLV made the survey of the ambiguities and incompleteness and, then, of the number of questions that could not be answered in a complete and correct way more difficult in CLV than in SND. Even if the different number of questions with incomplete or ambiguous answers for the two process components is not too relevant, they could have weighted differently in the evaluation of the correctness and similitude rate.

It was decided to evaluate the correctness and similitude rate for both the approaches and SND and CLV by considering only the questions having answers well defined. Figure 6 shows the scenario contributed to increase the correctness values when the questions with ambiguous or undefined answers

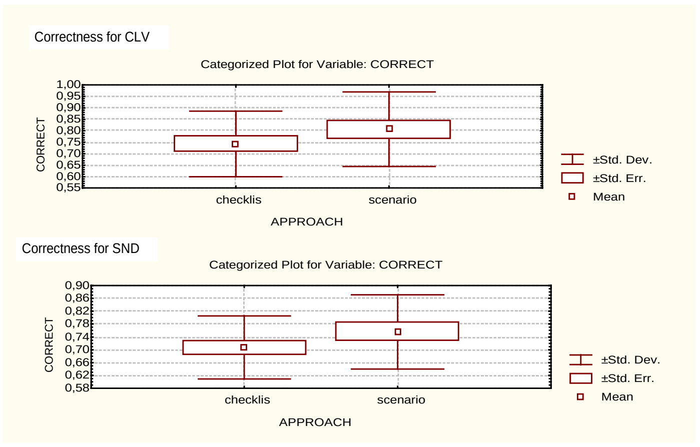


Figure 6 – Correctness obtained for the answers to the unambiguous questions

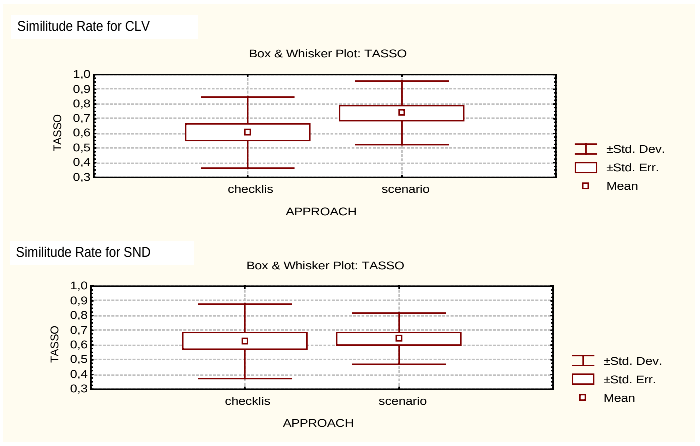


Figure 7 - Similitude Rate of the answers to the unambiguous questions

were not considered. Moreover, the correctness increases more for the most complex process component, CLV.

Analogous results were obtained for the Similitude Rate. The discharge of the ambiguous questions caused the reverse of the results in the Similitude Rate respect those previously obtained. Figure 7 shows that the best results were obtained for the most complex process component, CLV, where the similitude rate increases considerably compared to the one of the checklist.

4. Lessons learned and further work

The paper has presented the approach REP for analysing, evaluating and exploiting innovative process components. The experimentation has shown that a good level of correctness in the evaluation can be obtained by using this approach. The training to the use of the framework is the most important variable to obtain good results.

The results obtained showed that the reading from an informal description to construct an applicable formalism presents problems of correctness and completeness that are not easily solvable by using techniques based on scenario. In particular, the experience showed that the formalisation level of the scenario required more than the time foreseen for its full application. When the process component to be analysed were too complex, a major effort was required to understand its indefiniteness and ambiguities. It followed that the difficulties connected with the application of the scenario increased and the answers were randomly given and, then, not always correct. Besides, the complexity of the scenario induced to an unconscious utilization of the approach based on the checklist: when an experimental subject met difficulties, he used the checklist pretending to use the scenario.

This observation encouraged analysing the correctness and the similitude rate after having discharged the question with either ambiguous or indefinite answers. This caused better results and an increasing of the correctness values for the approach using the scenario.

This experience brought to the following two lessons learned:

1. the more difficult the comprehension of the informal description of the process component, the more effective the scenario;

2. it can happen that a step of the scenario does not find a univocal answer because of either the ambiguity or the lack of the information published. The scenario should foresee this situation and require a precise answer only when the information recorded is clearly described.

These lessons learned suggest some shrewdness that has to be adopted when of a scenario for transforming informal to formal description is defined. First of all, the scenario has to be clear, simple and to guide the user in a not ambiguous way. Moreover, it has to foresee that the answers can acquire the values *Ambiguous* or *Not existent* when the documentation is ambiguous or lack information. By taking into account the above lessons learned, the authors of the paper are working at an improvement of the scenario that considers the indication given above. The new scenario will be experimented with other experimental subjects. In any case, the experimentation needs to be replicated with a larger number of experimental subjects and, if possible, in industrial environments. To find industries willing to replicate the experimentation, it is necessary to assure them of its validity and inform them about the costs. Therefore, before asking industries for collaboration, the experiment needs to acquire maturity using students as experimental subjects.

The authors are ready to provide the experimental package and their assistance to those who would like to replicate the experimentation.

Bibliography

[1] Basili V. R., Daskalantonakis M. K., Yacobellis R. H., **Technology Transfer at Motorala,** *IEEE Software*, Vol. 11, No. 2, March 1994, pp.70-76

[2] Basili V.R., Rombach H.D., **Tailoring the Software Process to Project Goals and Environment**, *Proc. of the 9th International Conference on Software Engineering ICSE '87, Monterey, CA*, March 30 - April 2 1987, ACM press, pp. 345-357

[3] Cutillo, Lanubile F., Visaggio G., **Extracting application domain functions from old code: a real experience**, *Proc. of 2th Workshop on Program Comprehension WPC '93*, Capri, Italy, 8-9 July 1993, IEEE Comp. Soc. Press, pp 186-192

[4] Cimitile A., Visaggio G., **A Formalism for Structured Planning of a Software Project**, *Int. Journal of Software Engineering and Knowledge Engineering*, Vol. 4, No. 2, June 1994, World Scientific Publishing, pp. 277-300

[5] d'Inverno M., Justo G. R. R., Howells P., **A Formal Framework For Specifying Design Methods**, *SOFTWARE PROCESS - Improvement and Practice*, John Wiley & Sons, Vol. 2, 1996, pp. 181-195

[6] Fusaro P., Tortorella M., Visaggio G., **CREP – characteRising and Exploiting Process component: results of exeprimentation**, *Proc. of Working*

Conference on Reverse Engineering 1998, Honolulu, Hawaii, USA, October 12-14, 1998, IEEE Comp. Soc. Press

[7] Liu S., Wilde N., **Identifying objects in a conventional procedural language: an example of data design recovery,** *Proc. of Conference on Software Maintenance*, San Diego, CA, USA, 26-29 November, 1990, IEEE Comp. Soc. Press, pp. 266-271

[8] Sneed H., Nyáry E., **Migration of Procedurally Oriented COBOL Programs in an Object-Oriented Architecture**, *Proc. of 2th Working Conference on Reverse Engineering,* Toronto, Canada, July 1995, IEEE Comp. Soc., pp 217-226[12]

[9] Paulk M., Weber C., Curtis B., Chrissis M. B., **Capability Maturity Model. Guideline for Improving the Software Process**, *Addison Wesley*, 1994

[10] Song X., Osterweil L. J., **Comparing Design Methodologies Through Process Modelling**, *Proc. of 1st Int. Conference on Software Process, Redondo Beach, Ca*, October 1991, IEEE Comp. Soc. Press, pp. 29-44;

[11] Song X., Osterweil L. J., **Experience with an Approach to Compare Software Design Methodologies**, *IEEE Transaction on Software Engineering*, Vol. 20, No. 5, May 1994, pp. 364-384.

[12] Song X., **Systematic Integration of Design Methods**, *IEEE Software,* Vol. 14, No. 2, March-April 1997, pp. 107-117;

[13] Tortorella M., Visaggio G., **CREP - Characterising Reverse Engineering Process components methodology**, *Proc. of International Conference on Software Maintenance*, Bari, Italy, October 1 - 3, 1997, IEEE Comp. Soc. Press, pp. 222-231;

[14] Wohlwend H. and Rosenbaum S., **Schlumberger's software improvement program**; *IEEE Transaction on Software Engineering*, Vol. 20, No. 11, November 1994, pp.833-839.

SESSION 8:

New Proposals in Software Process Improvement

Expanding Goal Setting Theory Concepts – Using Goal Commitment Measurements to Improve Chances for Success in SPI

Pekka Abrahamsson
University of Oulu, Department of Information Processing
Science, P.O.Box 3000, FIN-90401 Oulu, FINLAND.
E-mail: Pekka.Abrahamsson@oulu.fi

Abstract

SPI managers and staff alike are troubled about clarifying the mechanisms that can sustain or impede the success of a software process improvement (SPI) project. This paper describes early results from an ongoing study aimed at improving the chances of success in SPI. It suggests that the chances can be improved by creating a better understanding of the process users' perception of the upcoming SPI initiative. The paper suggests the use of goal commitment measurement, adopted from the literature on goal setting, to discover these perceptions and to provide a means of communication between process users and SPI managers. A framework is developed for providing an operational definition of the attitude-based commitment model. The paper argues that the higher the level of the process users' goal commitment is, the better are the chances that the SPI initiative will be successful. Results from five interviews with SPI professionals about the proposed attitude-based model are reported, together with an empirical test conducted in 11 software engineering projects. These show that the components that form the goal commitment concept are essential to the success of a SPI initiative, and that the level of goal commitment could serve as an indicator of whether the goal of the SPI project will be achieved.

Introduction

Software is playing an ever-increasing role in today's society and in industry. As an answer to business and market needs, organizations have started to undertake software process improvement (SPI) initiatives aimed at increasing the maturity of their software processes (Humphrey 1989). While there are several approaches available for improving the software process, all of them share a common "requirement" in order to be successful – a commitment to SPI from all levels of the organization. Indeed, the significance of commitment for improving software processes has been widely recognized in the software process community both in the literature (e.g. Grady 1997; Humphrey 1989) and in articles concerned especially with the risks involved in SPI initiatives (e.g. Wiegers 1998; Statz, Oxley & O'Toole 1997).

As the discussion over commitment or the lack of it evolves, new fruitful ideas are needed to provide useful knowledge for both researchers and practitioners in the software process improvement field. One such idea is to harness the concepts of goal setting theory to the early phases of a SPI initiative. Goal setting theory is regarded as being among the most scientifically valid and useful theories in organizational science (Locke et al. 1988) and has become established as one of the most consistent and successful models of work performance (Earley, Connolly & Ekegren 1989). Goal setting theory is used in this paper as the basis for defining the concept of commitment.

The purpose of this paper is to report early results from an ongoing study aimed at improving the chances of success in SPI by increasing the awareness of SPI managers and staff of the process users' perceptions of the intended improvement initiative. Awareness can be raised by applying a measurement instrument that captures such important elements of the intended SPI project as process users' acceptance, effort expenditure and persistence regarding the defined goal. These elements are included in the concept of commitment introduced in the goal setting literature. A framework for translating the results is proposed, along with some views on how to construct a meaningful goal.

The attitude-based commitment model (Goal Commitment Questionnaire and Framework for Attitude-based Commitment) was tested by conducting five semi-structured interviews with SPI professionals who all had a strong background in improving software processes. All the professionals interviewed were interested in testing the model in their projects. In Addition an empirical test was conducted in 11 software engineering projects to find out whether there is any correlation between the level of goal commitment demonstrated and the success of the project. Even though the significant correlation remains to be proved there seems to be a tendency for a higher the level of goal commitment to lead to better performance in the project.

Background

The concept of goal setting

Goal setting falls within the domain of cognitive psychology (Locke et al. 1981) and is a widely used motivational technique (Locke 1975) that has been consistently supported by experimental research in both laboratory and field settings (e.g. DeShon & Landis 1997; Campion & Lord 1982).

The idea of assigning employees a specific amount of work to be accomplished – a specific task, a quota, a performance standard, an objective, or a deadline – is not new; as the concept of task was one of the cornerstones of scientific management, founded by Frederic W. Taylor more than 90 years ago (Latham & Locke 1979). The basic assumption of goal setting research is that goals are immediate regulators of human action. There is no assumption of a one-to-one correspondence between goals and action, however, since people may make errors or may lack the ability to attain their goal (Locke et al. 1981).

On the other hand, the effectiveness of goal setting does presuppose the existence of commitment. Locke et al. (1988) state that "*it is virtually axiomatic that if there is no commitment to goals, then goal setting does not work*". This paper focuses on measuring this commitment and using both the results and the process of measuring itself to promote SPI initiatives and enhance communication between the various stakeholders.

Defining goal commitment

Commitment is not easy to define because many different interpretations exist. Morrow (1983) identified 25 commitment-related constructs in the literature. After 40 years of research no consensus has been achieved over the general definition. According to Kiesler (1971), there are two main requirements for a useful definition of a psychological term. The literary definition should be clear and precise; the words should mean something, but this is not enough for research purposes. There must also be an operational definition, specifying the set of operations which define the concept so that the implications of theoretical discussions can be tested.

A distinction is made in commitment research between an "attitudinal" and a "behavioral" approach to commitment (Brown 1996, DeCotiis & Summers 1987, Mowday et al. 1979). In an organizational context, "attitudinal commitment" is classically viewed as "*the relative strength of an individual's identification with and involvement in a particular organization. Conceptually, it can be characterized by at least three factors: a) a strong belief in and acceptance of the organization's goals and values; b) a willingness to exert considerable effort on behalf of the organization; and c) a strong desire to maintain membership in the organization." (Mowday et al. 1982, 27).* Taking this to the individual level, attitudinal definition of commitment suggests that attitudes are affecting the way an individual will behave. On the other hand, the behavioral approach views commitment as a state of being, in which an individual becomes bound by his actions and through these actions to beliefs that sustain the activities and his own involvement (Salancik 1977). According to the behavioral approach, changes in the attitude are assumed to be consequences of changes in the behavior (Taylor 1994).

Goal commitment has been defined in goal setting research, in accordance with the attitudinal approach, as the degree to which the individual a) considers the goal to be important (acceptance), b) is determined to reach it by expanding her effort over time (effort expenditure), and c) is unwilling to abandon or lower the goal when confronted with setbacks and negative feedback (persistence) (DeShon & Landis 1997). According to DeShon and Landis, this definition reflects the most common themes used to describe goal commitment in the recent literature.

Having introduced the underlying theory and constructed the literary definition of commitment, the following chapters will concentrate on operationalizing this definition first by introducing an instrument (Goal Commitment Questionnaire), which can be used to capture the level of commitment, and secondly by introducing a framework (Framework for Attitude-based Commitment) that allows the interpretation of the results.

Measuring attitude-based commitment to SPI initiatives

A measurement model that is based on the above attitudinal approach to commitment now introduced. The basis for the attitudinal model is drawn from the goal setting literature. The underlying idea suggests that attitudes affect behavior in a clearly defined scope. The focus of commitment in this paper is SPI action or SPI goal[22] (e.g. if the primary goal is to improve software quality by reducing the number of defects in the development phase, then an appropriate goal could be 'establishing a review process'). When the goal is too vaguely defined or beyond the individual's scope, it becomes too abstract to consider[23]. In software engineering, Gilb (1988) emphasized the importance of clearly defined goals (as opposed to vaguely defined ones) and stated that "*projects without clear goals will not achieve their goals clearly*" (p.28).

It is clear in the goal setting literature that specific, difficult goals have a positive effect on performance (Dodd & Anderson 1996; Earley et al. 1989). It is claimed that in theory goal commitment moderates the relationship of goal difficulty to performance (Hollenbeck et al. 1989b). Originally it was proposed that goal commitment applies at the individual level (Locke 1968), but later it was established that similar effects on goal setting theory also apply at the group level (Klein & Mulvey 1995)[24], as the goal of a software process improvement project is to enhance process capability at both the individual and group level. Therefore, when goals in software process improvement initiatives are viewed as

---

[22] In the original goal setting research context the goal was defined as what an individual is trying to accomplish; being the object or aim of an action (Locke et al. 1981).

[23] A person might have a positive attitude towards preserving the nature (goal) but still does not recycle trash (behavior). Better results would be obtained by asking about her attitude towards recycling trash (Robbins 1993).

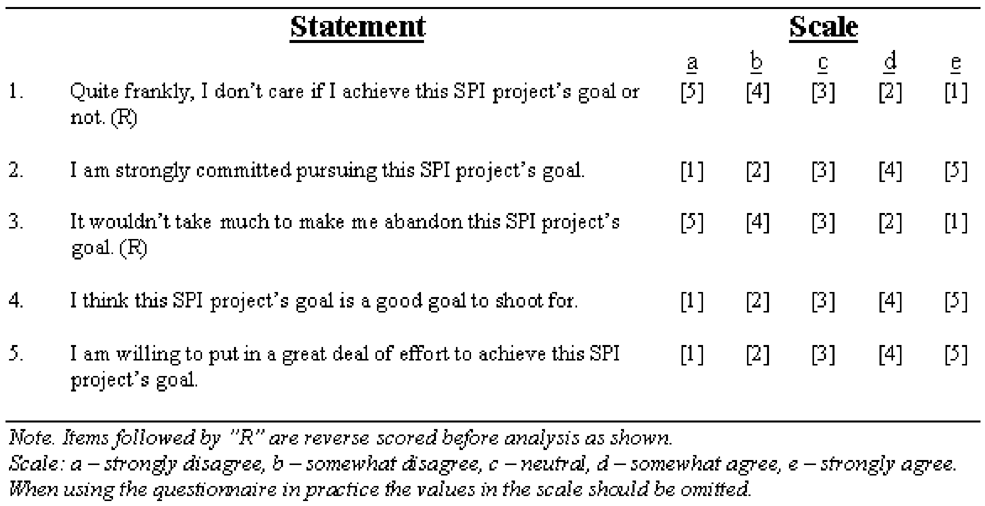
[24] Klein and Mulvey (1995) found a close relation between group commitment to goals and group performance, regardless of whether the goal was self-set or assigned.

challenging and the persons performing the improvements are committed to achieving the goals, the project has better chances of being successful.

Instrument for measuring commitment

There are several self-report scales for measuring commitment available in the goal setting literature. The most widely used instrument in many research areas is a nine-item scale developed by Hollenbeck, Williams and Klein (1989b; hereafter referred to as HWK). Large-scale use of the HWK scale is supported by the evidence that it is unidimensional (Hollenbeck et al. 1989a), the responses are stable over time (Hollenbeck et al. 1989b) and the scale is related to other significant constructs such as performance and motivational force (Hollenbeck et al. 1989a). The revised[25] HWK scale is presented in Table 1.

Table 1. Goal Commitment Questionnaire



	Statement	**Scale**				
		a	b	c	d	e
1.	Quite frankly, I don't care if I achieve this SPI project's goal or not. (R)	[5]	[4]	[3]	[2]	[1]
2.	I am strongly committed pursuing this SPI project's goal.	[1]	[2]	[3]	[4]	[5]
3.	It wouldn't take much to make me abandon this SPI project's goal. (R)	[5]	[4]	[3]	[2]	[1]
4.	I think this SPI project's goal is a good goal to shoot for.	[1]	[2]	[3]	[4]	[5]
5.	I am willing to put in a great deal of effort to achieve this SPI project's goal.	[1]	[2]	[3]	[4]	[5]

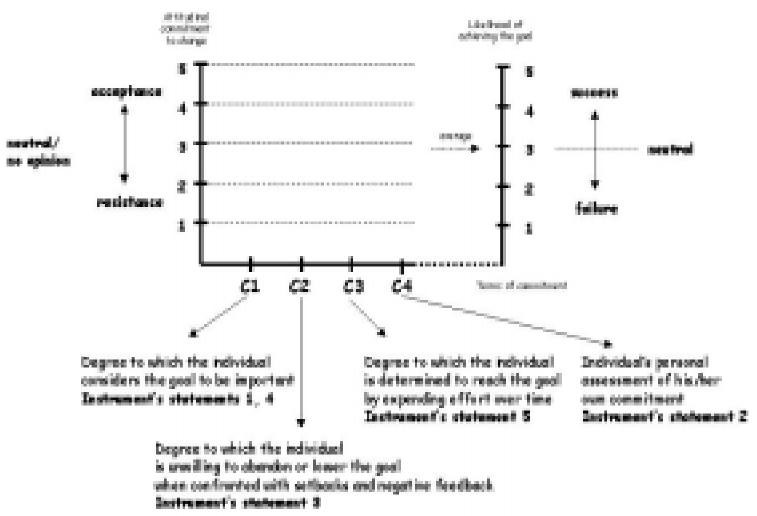
Note. Items followed by "R" are reverse scored before analysis as shown.
Scale: a – strongly disagree, b – somewhat disagree, c – neutral, d – somewhat agree, e – strongly agree.
When using the questionnaire in practice the values in the scale should be omitted.

[25] The questionnaire in Table 1 is revised in two ways: Firstly the wording of the items is changed to correspond to the focus of the paper and secondly four items are excluded from the scale in the light of later research (e.g. DeShon & Landis 1997; Tubbs & Dahl 1991)

Framework for interpreting the results

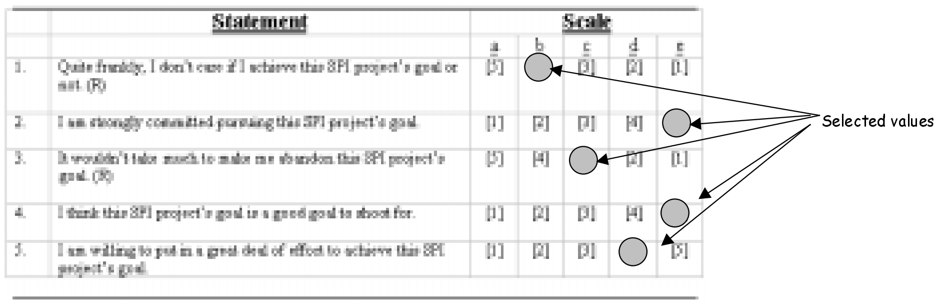
The framework for interpreting the results (Figure 1) consists of two parts: terms of commitment (components) and the explaining factor (the answer to the question: what does the result suggest?). The components are derived from the definition of goal commitment suggested by DeShon and Landis (1997).

The values for the components are calculated from the revised HWK scale as shown in Figure 2. Even though it is possible to calculate values for the components itself, the scale in its present form has too few items per component for it to be analyzed and conclusions drawn. The author therefore suggests that it should be applied as a whole to show the value of goal commitment.



Attitudinal commitment to change
Likelihood of achieving the goal
5
4
3
2
1
acceptance
neutral/
no opinion
resistance
average
success
neutral
failure
C1
C2
C3
C4
Terms of commitment
Degree to which the individual considers the goal to be important
Instrument's statements 1, 4
Degree to which the individual is unwilling to abandon or lower the goal when confronted with setbacks and negative feedback
Instrument's statement 3
Degree to which the individual is determined to reach the goal by expending effort over time
Instrument's statement 5
Individual's personal assessment of his/her own commitment
Instrument's statement 2

Figure 1. Framework for Attitude-based Commitment



	Statement	Scale				
		a	b	c	d	e
1.	Quite frankly, I don't care if I achieve this SPI project's goal or not. (R)	[5]	●	[3]	[2]	[1]
2.	I am strongly committed pursuing this SPI project's goal.	[1]	[2]	[3]	[4]	●
3.	It wouldn't take much to make me abandon this SPI project's goal. (R)	[5]	[4]	●	[2]	[1]
4.	I think this SPI project's goal is a good goal to shoot for.	[1]	[2]	[3]	[4]	●
5.	I am willing to put in a great deal of effort to achieve this SPI project's goal.	[1]	[2]	[3]	●	[5]

C1 score: (Statement 1 + Statement 4) / 2 = (4 + 5) / 2 = 4.5

C2 score: Statement 5 = 5

C3 score: Statement 3 = 3

C4 score: Statement 5 = 5

Score for goal commitment: (Statement 1 + 2 + 3 + 4 + 5) = (4 + 5 + 5 + 3 + 5) / 5 = 4.4

Figure 2. Example of evaluating scores

Even though it would be tempting to follow the current trend in goal setting research and claim that the success of the project could be foreseen by means of the model, this would not be true. The success of a software process improvement project depends on various factors, such as organization's culture, organization's management, process users, budgeting, scheduling and planning, the SPI project management, the SPI development environment and the SPI staff itself (Statz, Oxley & O'Toole 1997). A person does not have the same level of commitment all the time during a SPI project.. According to Brown (1996), a person evaluates his/her own commitment from time to time (once or many times) and the evaluation process itself is affected by current attitudes and circumstances, organizational factors and the "history" of the commitment – its development process, the reasons driving this development. Brown goes on to note that together these forces affect the way in which a commitment is evaluated and acted upon. The hypothesis underlying the attitude-based commitment model can thus be formulated as follows:

1) The level of goal commitment demonstrated by the process users is *one* of the factors affecting the success of a SPI project, and

2) serves as an indicator of whether the goal will be achieved.

Applying the attitudinal model in a SPI project

When applying the model, organization's culture should be taken into consideration, since the scale assesses people and their views and feelings about the SPI project. Some people might even find it unsuitable to evaluate their level of commitment if they feel they are working hard to achieve a certain goal. It is suggested that timing is the key issue here, as it is also in the SPI project itself. The questionnaire could be used at the very beginning of the SPI project – after the process users have been introduced to its goals. At this point they know what the project is supposed to achieve, but do not know exactly how to get there. The questionnaire[26] is handed out by the change agent or project leader and filled in anonymously. If the results show a low level of goal commitment, various conclusions can be drawn: that a) the process users do not see the project as important, b) the goal is not clearly focused on their needs, c) the goal is trivial, etc. The results should be presented to the group after the assessment is complete.

The purpose of the attitude-based commitment model is to serve as a <u>means of communication</u> between the SPI management, the SPI staff and the process users and as a <u>tool for the managers</u> to facilitate the change process. Another major purpose of the model is to <u>build up the awareness</u> of the SPI management of the components that form the foundation of commitment. This knowledge can be used implicitly by the change agent when discussing the SPI initiative with various stakeholders.

Small-scale testing of the attitude-based commitment model

The attitude-based commitment model was tested in a course held at the University of Oulu, in order to examine whether a) goal commitment correlates

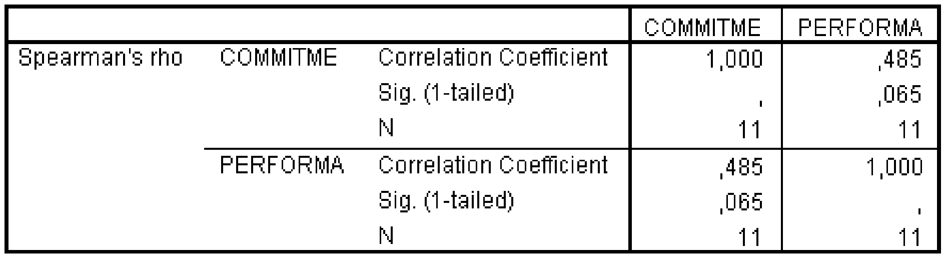
[26] The questionnaire can be modified so that it corresponds to the current goal of the SPI project.

with performance, and b) the model would work as a basis for a discussion forum. All 41 students that participated in the course had completed a software engineering project as members of a project team. The total number of project teams was 11, each having one to six students. The purpose of the course was for each project team to present the 'lessons learned' from their projects. The attitude-based commitment model was used as a tool for generating discussion and for analyzing the projects. The literature review conducted by Locke et al. (1988) suggests that there is no difference in the results between measurement of before and after performance. This supports the use of the attitudinal model in this case.

Students were given the task of evaluating first how well they managed to accomplish the goals set at the beginning of the project, on a scale from 1 (goals not achieved) to 5 (goals achieved well). A university representative evaluated the projects on the same scale. The students also evaluated the level of difficulty of the project on a scale from 1 (easy) to 5 (hard), since goal commitment has been thought in theory to moderate the relationship of goal difficulty to performance, in that performance is high only when both the goal level and goal commitment are high (Hollenbeck et al. 1989b). Lastly, the students filled in the goal commitment questionnaire and calculated the results, following the instructions given.

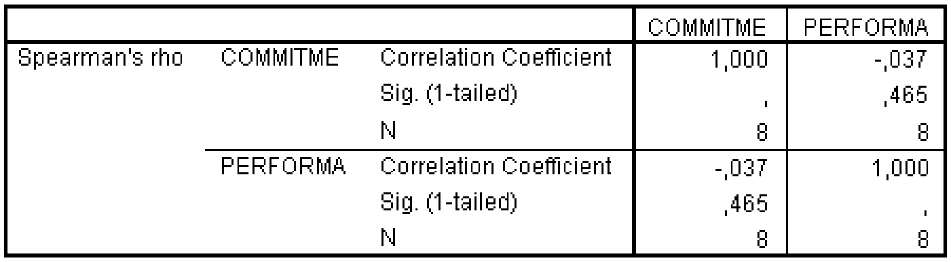
In order to determine the statistical correlation between the level of goal commitment and performance, Spearman's rho was calculated for the whole body of data (Table 2) and for the projects that were evaluated as difficult and challenging ones (Table 3).

Table 2. Spearman's Correlation Coefficient for the whole data set



			COMMITME	PERFORMA
Spearman's rho	COMMITME	Correlation Coefficient	1,000	,485
		Sig. (1-tailed)	,	,065
		N	11	11
	PERFORMA	Correlation Coefficient	,485	1,000
		Sig. (1-tailed)	,065	,
		N	11	11

Table 3. Spearman's Correlation Coefficient for difficult goals



			COMMITME	PERFORMA
Spearman's rho	COMMITME	Correlation Coefficient	1,000	-,037
		Sig. (1-tailed)	,	,465
		N	8	8
	PERFORMA	Correlation Coefficient	-,037	1,000
		Sig. (1-tailed)	,465	,
		N	8	8

Neither of the results (Tables 2 and 3) suggest that commitment is especially closely correlated performance. The sample of 11 projects however is nevertheless too small for any meaningful conclusions to be drawn. In order to evaluate whether goal commitment acts as an indicator of success, it may be useful to study following graphs (Figures 3 and 4).

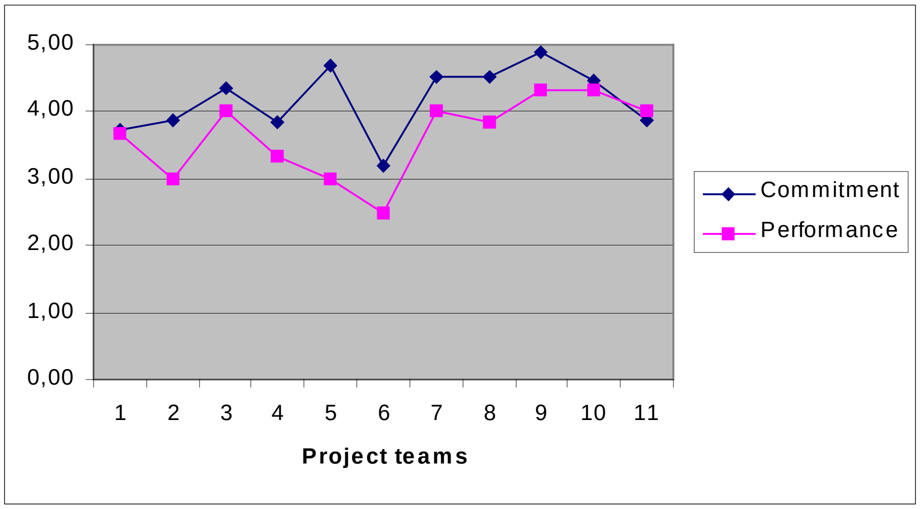


Figure 3. Visual comparison of goal commitment vs. performance

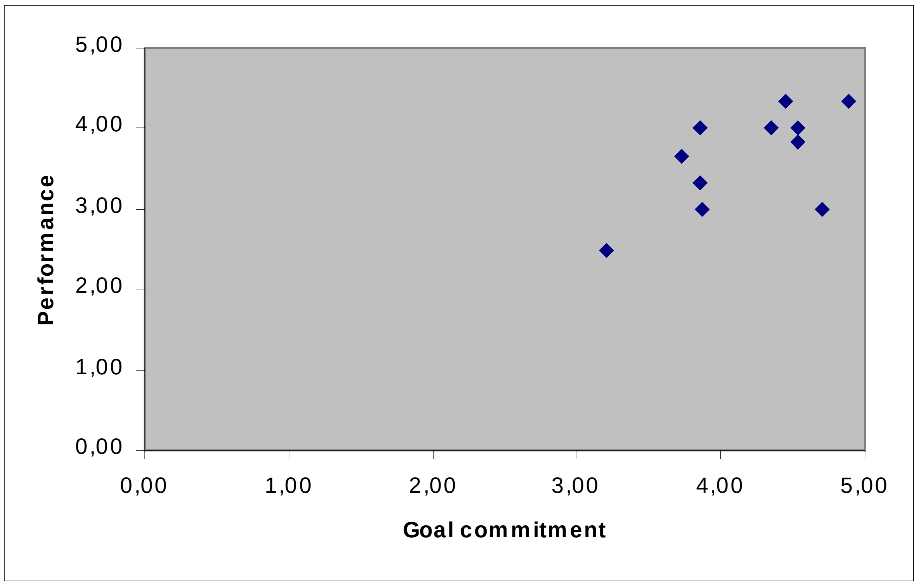


Figure 4. Scatter plot of goal commitment vs. performance

When examining the results presented in Figures 3 and 4, goal commitment may be viewed as an indicator of whether the goal will be achieved, even though the correlation was not statistically significant. The students felt that the results ‘seemed’ correct.

Usa of the attitude-based commitment model stimulated the discussion to a point that might not have been reached without it. This was especially the case when there was a significant difference between the levels of goal commitment and performance (e.g. project no 5 in Figure 3). The feedback received in these cases provided useful information such as the need to make sure that adequate resources are allocated for the project group *before* the project officially starts, and that feedback is given to the project group *regularly* as the project evolves, etc. Although the professionals interviewed had some concerns about the sensitivity of its use, the students did not feel the questionnaire to be too evaluative. This may be related to the fact that the test was conducted after the project.

Views from the field SPI professionals’ opinions

In order to test the attitude-based commitment model qualitatively, five semi-structured interviews were conducted in September – October 1998. All the people interviewed had considerable experience in leading software process improvement projects. The purpose of the interview was to find out a) whether the attitudinal model of commitment is relevant to SPI projects, b) where it could be possibly used, c) what might be the reaction of the group to be assessed when the questionnaire is administered, and d) how willing professionals are to try out the model in practice.

The results of the interviews suggest that all the components (C1, C2, C3 and C4 in Figure 1) are relevant to successful implementation of a SPI project, but that use of the model requires sensitivity to and knowledge of organization’s culture. From the viewpoint of professionals, the model could serve as a forum for discussion between stakeholders, since in professionals’ opinion the lack of communication is a major obstacle that has to be overcome in SPI initiatives. They suggested some improvements to the model, such as increasing the number of items in the questionnaire in order to achieve better reliability for the results and to allow an opportunity to analyze separate components as well. Another encouraging finding in the interviews was that all the professionals were willing to test the model in practice.

Conclusions

This paper has described the development and early results of an ongoing study aimed at improving chances of success in SPI by increasing the awareness of the SPI managers and staff of the process users' perceptions regarding the improvement initiative. This can be accomplished by measuring how committed the process users are to reaching the goal defined by the SPI managers and staff. By doing this, the SPI managers can place themselves in a better position for appreciating how well the process users accept the goal, and are willing to expend effort and persistance to achieve it. The knowledge acquired can be used to promote communication between various stakeholders involved in the SPI initiative.

The measurement instrument (Goal Commitment Questionnaire in Table 1) is based on the definition of goal commitment provided by DeShon & Landis (1997) and Hollenbeck et al. (1989b) and modified to suit the needs of software process improvement. In addition, a framework for interpreting the results (Framework for Attitude-based Commitment in Figure 1) was developed in order to have an operational model for attitudinal commitment.

The hypothesis that the goal commitment demonstrated by the process users is one of factors affecting the success of a SPI project was supported by the results of the interviews as the professionals agreed that all the components presented in the framework occupy significant roles in a SPI project. The second hypothesis suggested that the level of goal commitment would serve as an indicator of whether the goal will be achieved. There is not enough evidence to support or reject this hypothesis according to the data acquired in the empirical test presented in this paper. Limitations to the empirical test were the small number of cases (11) and the fact that the projects analyzed were not SPI projects but software engineering projects. Ten projects reported that they had had to adopt new procedures and technologies, however, which makes them somewhat similar to SPI projects.

The positive feedback received from the professionals demonstrated that there is a need in the SPI community to include a human element in models that guide the improvement process. This includes creating a better understanding of such complex processes as motivating people and committing them to organizational change. This paper provides a new insight into viewing commitment not only as a psychological state but also as a combination of influential elements that make people willing to pursue a goal.

References

Brown, R. B. 1996. Organizational commitment: clarifying the concept and simplifying the existing construct typology, in Journal of Vocational Behavior, Vol. 49, 230-251.

Campion, M. A.. & Lord, R. G. 1982. A control systems conceptualization of the goal-setting and changing process. Organizational Behavior and Human Performance, Vol. 30, pp. 265-287.

DeCotiis, T. A., & Summers, T. P. 1987. A Path-Analysis of a Model of the Antecedents and Consequences of Organizational Commitment. Human Relations, Vol. 40, No. 7, pp. 445-470.

DeShon, R. P. & Landis, R. S. 1997. The Dimensionality of the Hollenbeck, Williams, and Klein (1989) Measure of Goal Commitment on Complex Tasks. In Organizational Behavior and Human Decision Processes. Vol. 70, No. 2, pp. 105-116.

Dodd, N. G., & Anderson, K. S. 1996. A test of goal commitment as a moderator of the relationship between goal level and performance. Journal of Social Behavior and Personality, Vol. 11, No. 2, pp. 329-336.

Earley, P. C., Connolly, T., & Ekegren, G. 1989. Goals, strategy development, and task performance: Some limits on the efficacy of goal setting. Journal of Applied Psychology, Vol. 74, No.1, pp. 24-33.

Gilb, T. 1988. Principles of Software Engineering Management, Wokingham, Addison-Wesley.

Grady, R. B. 1997. Successful Software Process Improvement. Prentice-Hall, NJ.

Hollenbeck, J. R., Klein, H. J., O'Leary, Am. M., & Wright, P. M. 1989a. Investigation of the construct validity of a self-report measure of goal commitment. Journal of Applied Psychology, Vol. 74, No. 6, pp. 951-956.

Hollenbeck, J. R., Williams, C. R., & Klein, H. J. 1989b. An empirical examination of the antecedents of commitment to difficult goals. Journal of Applied Psychology, Vol. 74, No. 1, pp. 18-23.

Humphrey, W. S. 1989. Managing the Software Process. Addison-Wesley

Kiesler C., A. 1971. The Psychology of Commitment: Experiments Linking Behavior to Belief, Academic Press.

Klein, H. J., & Mulvey, P. W. 1995. Two Investigations of the Relationships among Group Goals, Goal Commitment, Cohesion and Performance. Organizational Behavior and Human Decision Processes, Vol. 61, No. 1, pp. 44-53.

Latham, G. P., & Locke, E. A. 1979. Goal-setting – A Motivational Technique That Works. In Barry M. Staw (ed.), Psychological Dimensions of Organizational Behavior, Prentice Hall, 1995, 2nd ed.

Locke, E. A. 1968. Toward a theory of task motivation and incentives. Organizational Behavior and

Human Performance, Vol. 3, pp. 157-189.

Locke, E. A. 1975. Personnel attitudes and motivation, Annual Review of Psychology, Vol. 26, pp. 457-480.

Locke, E. A., Latham, G. P., & Erez, M. 1988. The determinants of goal commitment. Academy of Management Review, Vol. 13, No. 13, pp. 23-39.

Locke, E. A., Shaw, N. R., Saari, L. M., & Latham, G. P. 1981. Goal setting and task performance: 1969-1980. Psychological Bulletin, Vol. 90, No. 1, pp. 125-152.

Morrow, P. C. 1983. Concept redundancy in organizational research: The case of work commitment. Academy of Management Review, Vol. 8, No. 3, pp. 486-500.

Mowday, R. T., Porter, L. W., & Steers, R. M. (1982). Employee-organization linkages: The psychology of commitment, absenteeism, and turnover. New York: Academic Press.

Robbins, S. P. 1993. Organizational Behavior. Prentice-Hall, Englewood Cliffs, New Jersey, 6. ed.

Salancik G. R. 1977. Commitment is too Easy! In Tushman M & Moore W. (eds.), Readings in the Management of Innovation, pp.207-222. Boston: Pitman 1982

Statz, J., Oxley, D., & O'Toole, P. 1997. Identifying and Managing Risks for Software Process Improvement, CrossTalk, April, http://www.stsc.hill.af.mil/CrossTalk/1997/apr/Identifying.html

Taylor, W. A. 1994. Senior executives and ISO 9000 – Attitudes, behaviours and commitment. International Journal of Quality and Reliability Management. Vol. 12, No. 4, pp. 40-57.

Tubbs, M., & Dahl, J. G. 1991. An empirical comparison of self-report and discrepancy measures of goal-commitment. Journal of Applied Psychology, Vol. 76, No. 5, pp. 708-716.

Wiegers, K. E. 1998. Software Process Improvement: Eight Traps to Avoid. CrossTalk, The Journal of Defense Software Engineering. September.

TAILORING PROCESS IMPROVEMENT TO SMALL COMPANIES USING A METHODOLOGY FOCUSED ON THE INDIVIDUALS

Guido Cardino
Socrate Sistemi S.a.S
Via Sormano 2/4, I-17100,Savona, Italy
Email: gucardin@tin.it

Andrea Valerio
Dipartimento di Informatica, Sistemistica e Telematica
Università degli Studi di Genova
I-16145 Genova, Italy

Abstract

Software development is currently characterized by rapid advances in technology, growing complexity required by customers and strong market competition. Small companies often have not the resources to build the proper organizational framework needed to face these challenges. They can instead apply a bottom-up approach using strategies for personal process improvement. This article presents the ASPIDE project, a process improvement experiment funded by the European Community (Esprit ESSI Project 24331). The focus of the experiment is the introduction the Personal Software Process (PSP) improvement discipline in Socrate Sistemi, a small Italian software company mainly involved in the development of management system for fashion stores. This approach seems to be suitable for small enterprises needed to improve their overall development performances in order to obtain greater competitiveness. The available results show a positive impact on the corporate organization and on the competence of the employees, suggesting also solutions for possible problems.

Introduction

This work presents a case study on the adoption of the Personal Software Process (hereafter PSP) in a small software firm located in Italy named Socrate Sistemi. This experiment has been funded by the European Community in the context of the ESSI initiative as the “ASPIDE” Project 24331. Its purpose was to evaluate the benefits and possible drawbacks deriving from the introduction of a software process discipline focusing on individuals in a very small company in which it is really difficult to plan business process reengineering activities which require a strong organizational support. In this context, due to the characteristic approach of the PSP based on the decentralization of the improvement responsibility, this discipline seems to match the needs of small enterprises. The PSP aims to foster continuous improvement, using a bottom-up approach that begins from personnel. The results achieved show the positive impact of this approach both on personnel work and on the organization.

Case Study Description

Introduction to the Personal Software Process

The Personal Software Process (hereafter PSP) was developed by Watts Humphrey[1] working at the Software Engineering Institute. It is a defined and measured software process designed to be used by an individual software engineer in order to monitor and control his daily work in a structured way. Whereas it is intended mainly to the development of medium-sized software modules, it is adaptable to other personal tasks (like production of design specifications and software documentation).

Drawing its structure from the SEI Capability Maturity Model (CMM)[3] for Software, also the PSP is based on process improvement principles. The difference is that the CMM is focused on improving the overall organizational capability, whereas the focus of the PSP is the individual engineer. In this sense, the PSP is exactly the individual-focused counterpart of the CMM, which stresses organizational and company-wide issues. In the PSP framework, software engineers work using a disciplined, structured approach making them easier to plan, measure, and track their efforts, to manage their product quality, and to apply quantitative feedback to improve their personal development processes. The rise in competence of the engineers, which in this viewpoint play an active role in process improvement, on turn improve the organization’s capability, especially in the case of small companies.

The PSP (like the CMM) is organized into different layers. The layered structure of the PSP is shown in figure 1. Each level builds on the prior level by adding a few process steps to it. This minimizes the impact of process change on the engineer, who needs only to adapt the new techniques into an existing baseline of practices.

The baseline personal process (PSP0 and PSP0.1) establishes an initial foundation of historical size, time, and defect data. Development time, defects, and program size are measured and recorded on proper forms. The next steps, PSP1 and PSP1.1, focus on personal project management techniques. They introduce size and effort estimating, schedule planning, and schedule tracking methods.

PSP2 and PSP2.1 add quality management methods to the PSP: personal design and code reviews, a design notation, design templates, design verification techniques, and measures for managing process and product quality. The goal of quality management in the PSP is to find and remove all defects before the first compile. Two new process steps, design review and code review, are included at PSP2 to help engineers achieve this goal. These are personal reviews conducted by an engineer on his/her own design or code guided by personal review checklists derived from the engineer’s historical defect data. In this way, starting with PSP2, engineers begin using the historical data to plan for quality and control quality during development. According to some studies conducted by the Software Engineering Institute[2], with sufficient data and practice, engineers are capable of eliminating 60% to 70% of the defects they inject before their first compile.

The Cyclic Personal Process, PSP3, addresses the need to efficiently scale the PSP up to larger projects without sacrificing quality or productivity. Large programs are decomposed into parts for development and then integrated. This strategy ensures that engineers are working at their achieved optimum productivity and product quality levels, with only incremental, not exponential, increases in overhead for larger projects.

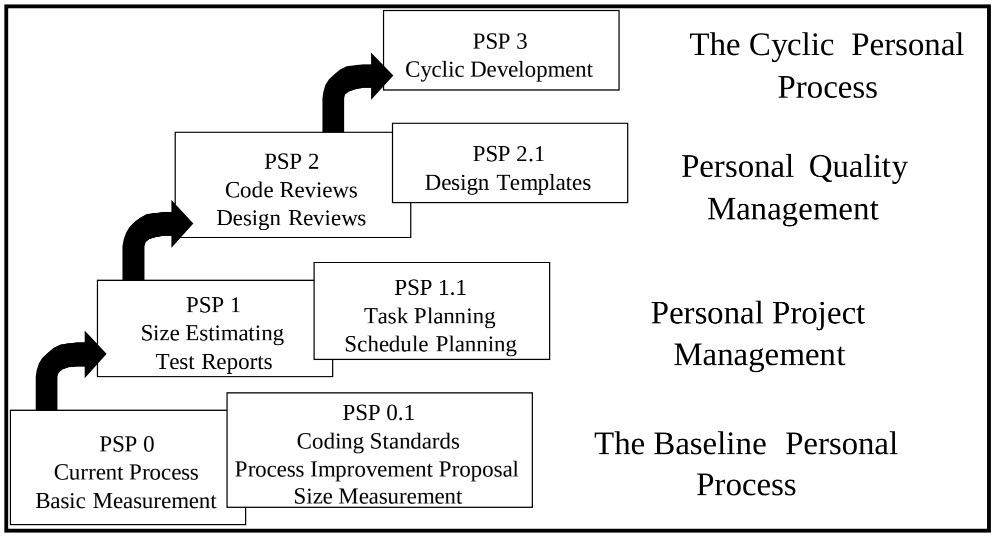


Figure 1: The PSP Levels

The ASPIDE Project

The ASPIDE ESSI Project 24331 is the application of the first PSP steps (PSP0 and PSP0.1) to the employees of a very small sized italian software, Socrate Sistemi S.a.S. Socrate Sistemi is a small company (about ten employees), with a sale organization composed of several external professionals that embraces all the parts of Italy. It is involved in the production of software for the administration of sale points and head offices especially in the area of fashion stores.

The business motivation of the experiment can be traced back to the company need to maintain a proper productivity and quality level even in a context characterized by new technologically rich requirements from the customers (e.g. integration of the Internet inside their company organization), by the intrinsic complexity of the new products and by the raise of costs and market competition. The resulting problems were handled with difficulty by the company because no measurement mechanism was used during the development: without a punctual knowledge of the performances of the development, it was impossible to put into practice even the simplest improvement step. Moreover, a critical constraint was the limited size of the company: considering the current available resources, it is impossible to reserve effort for an organizational support team.

For all these reasons, the definition of a measurement mechanism (following PSP0 and PSP0.1) was the first step in the planned business process reengineering strategy. In particular, the PSP has a de-centralized approach which does not need (at least after a period of introduction) a particular support. Rather, every employee is responsible for his own professional growth. From this perspective, the PSP is an interesting alternative to the process improvement processes based on a top-down approach, i.e. imposing a formal process model to the projects and personnel involved. It candidates to be the best choice for small and medium companies, where a bottom-up strategy beginning from the personnel could demonstrates to be the best approach. The steps for personal improvement are then organized in a company-wide way so that they ensure the corporate growth.

Case Study Scenario

Starting Technological Context

From a technological point of view, the experiment was applied to a project aimed to the complete re-engineering of a complex software product. This is a system for the administration of fashion, shoes, sportswear and clothing stores with the possibility of managing different points of sales in a distributed network. In addition, heterogeneous physical devices (such as portable optical bar-code readers) are integrated in a unique environment and statistical analysis on the sales can be performed. The starting old version of the system ran on SCO-UNIX workstations or DOS PC without a graphical user interface, and it was about 8KLOC of COBOL. The system has been ported to Microsoft Windows 95/98 and NT, and a complete re-design of the system was concurrently executed (e.g. to take into account Y2K and Euro problems). Senior analysts estimated that most of the modules will require a complete re-development due to great changes in the presentation of data or interaction with the user. The porting and reengineering project interested most programmers of Socrate Sistemi, constituting an adequate context for the case study.

Company Assessment

The starting scenario for the case study has been evaluated through an assessment of the company's profile and development capability. The context considered is that of a small company involved in the production of software for fashion stores. The maturity level of the organization, according to the SEI CMM process assessment scheme[3] is level 1, featured by a process aimed at the achievement of short term goals, with several deficiencies in the medium or long term and focused on people-driven processes. The main detected weaknesses was the following:

- No measurement program is defined for gathering data on the time a developer spends on each software task and for tracking the number of defects they introduce or remove in each process step. This problem has been directly addressed by our experiment.

- Configuration and version management lacks almost completely. This contributes in turn to decrease productivity and makes reuse practices and testing difficult. This problem will be probably faced in the short term, on the basis of productivity and quality records obtained by the PSP application.

- No mechanisms are used as support for the planning, so that it is difficult to understand if the project planning suffer from over-allocations or other flaws. Due to the small size of the company, projects often run in parallel and have to share human, hardware and software resources: project planning without supporting tools becomes very difficult in this situation. Moreover, the absence of a defined mechanism for the measurement of the developers' work makes very difficult to define plans based on time, efforts or size records. Socrate Sistemi has been evaluating the possibility to extend the PSP application to the next level (PSP1), supplying a bottom-up solution to this problem on the foundations laid by the ASPIDE experiment.

- Defined and planned testing activity lacks and the results are that many bugs and defects are detected by the users, decreasing their satisfaction. Moreover test activities and results are not formally recorded and tracked. The quality problem will be considered in the long run, after a proper solution of the previous deficiencies. On the other hand, from our viewpoint, a structured introduction of PSP3, organized at the company level, can give important feedback in this direction.

The ASPIDE experiment on turn was built upon the positive factors highlighted by the company assessment. The company has several competent and creative people, with a good mix of technical knowledge and the demonstrated capacity of finding new directions for the evolution of the products. Thanks to the limited size of the company, developers participate with the management in the definition of the improvement plan for the software process and in general of the future strategies. The environment is generally very good.

Senior engineers are already working to define sound development practices in order to discover bugs or defects as early as possible in the development cycle. The management is openly encouraging this process, fostering the proper involvement of all employees. Moreover, to maintain a strict connection with the customers' expectations the development process encourages the production of prototypes of the final application in order to identify and solve problems in the early stages of the process. As an additional positive point, the company assessment itself has contributed to augment people motivation and to increase the awareness for the need for process improvement. As a result, a positive attitude towards organizational improvement is featuring the company.

Case Study Evaluation

Technical Results

From a technical viewpoint, the experiment allowed the assessment of the productivity of the development process, highlighting important technical factors that affects the development performances. As an example, the analysis pointed into evidence that for the Windows-based application of the experiment (which is on turn a good representative of the system produced by the company) the main technical challenge is the definition the user interface. About 70% of the total effort have been used in the production of this part of the system. The developers involved in the graphical part had a substantially lower productivity than the others did: as regarding the delivered lines of source code, they produced an average of 250 LOC/day instead of about 700 LOC/day.

The analysis allowed by PSP data recording was important to establish that software reuse can play a substantial role in the company software development. Currently, developers exploit a unstructured reuse approach based on code reuse and only few reusable assets are published in a organized way at the company level. Even with this poor strategy, a great part of the system was developed exploiting the integration of already produced software modules (about 80% of the data management engine have been re-used with slight modifications from the past applications). The experiment raised the awareness that reuse can be a technical factor to reach outstanding performances. This aspect will probably be investigated in the future with other focused experiments.

Organizational Results

The integration of PSP0 and PSP0.1 in the engineers' activities allowed the re-structuring of the entire company's work organization to meet the requirements of the new kinds of products for the Windows operating environment.

The analysis evidenced that whereas the productivity of the individual developers is relatively stable for the development of data management procedures, it is not the same for the graphical user interface: some GUI modules, for instance, have been developed twice by different employees before reaching an acceptable level of quality. The main critical factor here is the creativity requested to developers for a successful design of dialogs, pictures and animations. To solve this problem, a new work organization has been implemented in the mid-term of the experiment and evaluated as a success in its second half. The work has been organized by competence, rather than subdividing among developers the functional modules as happened before; in this way the applications demanding particular skills are managed by specialized task forces of 1-2 developers, exploiting the personal competence and creativity pointed into evidence by the structured data recording fostered by PSP0 and PSP0.1.

As a small term organizational achievement in the light of quality control, even the simple process monitoring allowed by PSP0 and PSP0.1 was able to detect a flaw in the allocation of human resources that caused a product quality problem that could be seen by customers. In fact, due the stress on the graphical user interface and to the reuse of the data management part of the baseline system, most of the effort was allocated for the production and testing of the new code. Too few resources were allocated to the testing of the reused code. The analysis of the personal records showed this possible risk, and therefore a new testing activity was scheduled for the quality assurance of the reused part of the system. The results of this additional testing activity highlighted that small changes in the data caused some defects in the modules reused with slight changes, and these defects could have been detected after the release of the product by customers. From this point of view, the measurement framework enabled better quality procedures, fostering the identification of faults in the work organization. This is particularly important for a small company like Socrate Sistemi, since due to the low available resources it is impossible to formally define a quality supervision organization.

Human Issues

The PSP has a substantial impact on the individuals, fostering their professional growth and teaching a more rigorous approach in daily work. The ASPIDE final evaluation shows that the measurement activities requested by PSP0 and PSP0.1 have been integrated satisfactorily in their routine, even if, especially at the beginning, it was necessary to overcome some resistance. In fact, personnel generally considers measurement as a time wasting activity, since it does not produce any visible effect on their work but consumes their time. The planned initial training has demonstrated to be a valid support in this direction, but a continuous support during the first months of PSP application in daily work is fundamental to sustain the individual motivation and then to achieve the sought benefits in the long run. In the absence of such a support, the quality of the recorded data has been not satisfactorily at the beginning: if the developers see measurement only as an imposition, they tend to fill the forms using inappropriate or not complete information.

Despite the initial diffidence of developers towards data collection, it has been shown that after a start-up period the PSP can have a substantial impact on the competence of the software engineers. What turned out is that they gained the awareness that software development can be measured, understood and then improved. Considering some developers' initial way of working, in some aspects more similar to handcrafting rather than industrial production, the main achievement could be considered the awareness that software development can become a discipline, with formal procedures and a technique that can have a positive impact on the work. Moreover, if we take into account the skill required to manage a discipline like the PSP, some developers acquired a good understanding of the methods and techniques applied, even if some used tools (e.g. the statistical ones) are not a normal knowledge for an average programmer. This fact demonstrate that the achieved competence is not only a transitory benefit.

Key Lessons

The presented case study pointed into evidence key lessons that are valuable for small organizations, working in the field of software engineering without a defined and structured process, which are considering process improvement alternatives for facing the growing market pressure. The main lesson is that a supervision mechanism for the monitoring and control of the development process can be effectively put into practice even for those small companies on the basis of individual efforts, in a decentralized approach. In this respect, the PSP seems to be a feasible solution, especially because it is highly adaptable to specific corporate or personal needs.

Moreover, this approach can work even in organizations for which software development has been considered more a creative, almost artistic, work than an industrial practice. In this context, it has to be considered that a rigorous discipline for software engineering such as the PSP requires a substantial paradigm shift, and therefore a proper training and support activity can be the key for the success.

As a final conclusion, it has to be noted that apart from the achievements of the objectives of the different PSP levels, the discipline gives to the individual and to the company a knowledge of the development process which allows for the identification of the effects of all software-related practices, such as software reuse or the introduction of new tools.

References

[1] Watts S. Humprey, A Discipline for Software Engineering, SEI Series in Software Engineering, Addison-Wesley Publishing Company, Inc., 1995.

[2] Will Hayes, James W. Over, The Personal Software Process (PSP): An Empirical Study of the Impact of PSP on Individual Engineers, SEI/CMU Technical Report CMU/SEI-96-TR-001, December 1997.

[3] Paulk, M., Curtis, B., Chrissis, M., Weber, C., The Capability Maturity Model for Software (Version 1.1), Software Engineering Institute, Technical Report, 1993.

Moving Towards Modelling Oriented Software Process Engineering: A Shift from Descriptive to Prescriptive Process Modelling

Simo Rossi
Nokia Telecommunications
P.O. Box 55, FIN-40351 Jyväskylä, FINLAND
E-mail: simo.rossi@ntc.nokia.com, Tel: +358 14 5779 811

Abstract

This paper reports a non-technology-oriented approach to software process modelling, which is a significant and integral part of software process engineering. The methodical aspects, the diverse ways that modelling can take place, are emphasised while four software process modelling principles are listed and experience gained in applying these to prescriptive modelling is shared. We do not, however, report any specific modelling process or method since the focus is on empirical experience and related discussion. Associated with the reporting of modelling experience, this paper also describes an application of two CMM key process areas namely organisation process focus and organisation process definition dedicated to software process engineering and modelling.

Introduction

In pursuing towards a rational design process, process modelling offers an effective means for achieving understanding and facilitating communication of the software process by making it more visible and explicit. Thus, understanding is a prerequisite for an effective process improvement and management as well as for automating parts of the software process (Curtis et al., 1992; Heineman et al., 1994). The methodical aspects of software process modelling, however, have received scantly attention while many of the research efforts have aimed at developing technology support (e.g. process-centred software engineering environments) to define, evolve, analyse and execute process models (see, e.g., Finkelstein et al., 1994). In addition to process technology, process modelling languages (PMLs) have received a great deal of attention in the field of software process research (see, e.g., Shepard et al., 1992; Bandinelli et al., 1993; Deiters and Gruhn, 1994; Christie, 1993; Dutton, 1993; Kaiser et al., 1993; de Bunje et al., 1996). Besides technology support and PMLs, even few modelling methods have been introduced to facilitate management of the modelling process by bringing discipline to it (see Kellner and Hansen, 1989; Höltje et al. 1994; Klingler and Schwarting, 1994; Kontio, 1995). These methods mainly depict the modelling process in a life-cycle fashion describing what to do in different phases of the modelling. Most of the methods duly appreciate the understanding of the process context as well as flexibility in accommodating the modelling process to the requirements of the process context. However, the experience associated in the application of process modelling methods, languages and technology support, in process improvement has not been described thoroughly in literature.

In this paper we report a study investigating prescriptive software process modelling. The research issues discussed are "what effects the shift from descriptive to prescriptive modelling have on the modelling" and "are the four software process modelling principles, namely flexibility, PML engineering, use of multiple data sources and process actor participation applicable to prescriptive modelling". We have studied these questions by conducting a prescriptive modelling experiment in an industrial software development environment.

The rest of this paper is structured as follows. The next section introduces the four modelling principles providing theoretical background for the modelling experiment. Then, Section 3 provides a description of the software organisation in which the modelling took place. Section 4 continues by describing how the process modelling has previously been used to boost process engineering in the research context. Then, Section 5 reports the experience resulted from testing the four principles in a prescriptive modelling case. Finally, in Section 0 we draw conclusions from this study.

Principles hidden behind the curtain

Current software process literature lacks methods and guidance for descriptive process modelling. Especially the participation of process actors in the modelling process as well as engineering of methods and modelling languages in accordance to the process context have received very little attention, although they have been a popular research theme in the neighbouring information systems development field (see, e.g., Hirschheim and Klein, 1992). Therefore Rossi and Sillander have reported a study pertaining to fundamental software process modelling principles (1998a). This study resulted in four process modelling principles, proposing issues to be taken into account in modelling a software process in detail for the first time when the objective is to facilitate understanding and communication of the modelled process

The first principle, the principle of flexibility, emphasises on the contextual relationships of the modelling and calls for context-orientation throughout the modelling process. This principle is based on the observation that there exists no one right software process (see, e.g., Osterweil, 1987) or method for every development situation (see, e.g., Kumar and Welke, 1992). On the contrary, new ones need to be constantly developed and the existing ones modified for the different and changing development situations. This applies to the software process modelling processes and methods as well.

The second principle, the principle of PML engineering, is a tool for adapting the principle of flexibility to modelling and it calls for situational PML engineering based on the requirements of the context. We have heightened it as a separate principle owing to the central role that the PML plays in modelling. Thus, in addition to the flexible selection of the PML, this principle suggests that the selected language or languages should be modified to suit the needs and abilities of the process context. The idea of PML engineering is simply to apply the method engineering experience (Kumar and Welke, 1992) to software process modelling languages. This appears to be almost completely foreign to the current SPM methods and even to the process modelling research as a whole (Rossi and Sillander, 1998b; Koskinen and Marttiin, 1998).

The third principle, the principle of using multiple data sources, prompts to simultaneously exploit more than one data source in modelling. This principle results from the fact that most process modelling is currently based on data obtained from multiple data sources and the most popular sources of modelling data are process documentation, questionnaires and interviews.

The fourth and final principle, the principle of process actor participation, aims at involving the process actors themselves in the modelling process. This principle is to a high degree grounded on the participative design concept introduced in the field of information system development field (see, e.g., Hirschheim and Klein, 1992). It is difficult to imagine a more effective communication occasion than the actors and the modellers discussing and evaluating the details of the process model together.

Process context: PMR Terminals

We conducted our modelling study while working for Nokia Telecommunications PMR Terminals unit (Professional Mobile Radio) which develops and produces professional mobile radio terminals for government officials and public trunking operators (e.g. police radios). The development of the embedded software for these terminals vary from the end-user related user interface design supported by CASE tools and graphical design languages up to hardware related assembler programming. Moreover, the software development takes place in a multi-project environment in which three or more projects are running concurrently. The procedures, steering and resources for these projects are provided by the line organisation.

Because of the tightening competition, customers' demands, accelerating product development cycles and extensive growth, i.e. because of the internal and external pressure, PMR Terminals has recently invested more and more in software process engineering in order to improve the quality and performance of software development. First, resources have been allocated and infrastructure established for the process engineering responsibility, which is founded on Madhavji's process cycle (1991). Second sign of the investment on process engineering is the defined process engineering process depicted in Figure 1, which has been introduced into systematic and continuous use. This process is founded on such famous approaches as the Quality Improvement Paradigm (QIP) (see, e.g., Basili and Rombach, 1988) and the CMM (Capability Maturity

Model) (Paulk et al., 1995). Also a more recent approach named Helical model has contributed to the process (Kellner et al., 1996).

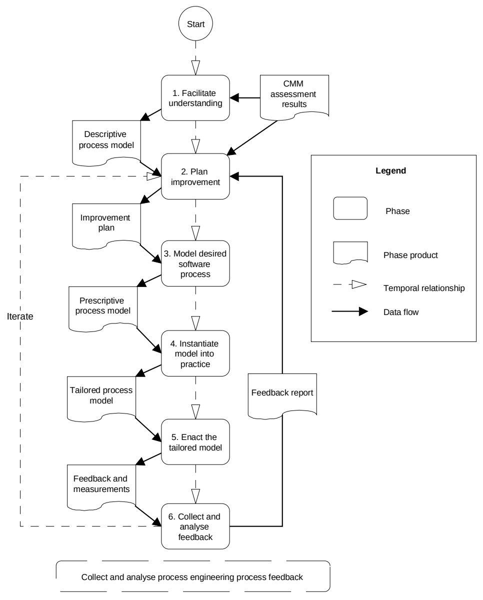

Start
1. Facilitate understanding
CMM assessment results
Descriptive process model
2. Plan improvement
Legend
Phase
Phase product
Temporal relationship
Data flow
Improvement plan
3. Model desired software process
Prescriptive process model
Iterate
4. Instantiate model into practice
Tailored process model
Feedback report
5. Enact the tailored model
Feedback and measurements
6. Collect and analyse feedback
Collect and analyse process engineering process feedback

Figure 1. The PMR Terminals Software Process Engineering Process

PMR Terminals process engineering process is supported with the CMM. The CMM has internally been used for assessing the maturity and capability of the software development as well as to support the improvement efforts for several years. From the CMM point of view, the focus of these improvement efforts has been on levels two and three. The steps of the process engineering process demonstrate the significance of the software process modelling in process engineering. Thus, each of the process engineering steps is crucial and would deserve more attention. However, in this paper we focus especially on step three, modelling a desired software process.

Process modelling in PMR Terminals

Until recently, software development and process management in PMR Terminals has been based on high-level models, which merely give some general guidelines and milestones for the product development. The lack of details in these models has caused the software development and project management to become very person-dependent. Thus, the success of the projects is very much up to the experience and capability of process actors and managers. Recent internal CMM assessments have indicated the very same shortcomings. Therefore, based on the assumption that a defined, documented, executed and measured process is a prerequisite for effective improvement (see, e.g., Kellner et al., 1996), a considerable share of the software development effort has been invested in process engineering boosted with process modelling recently (approximately 2% of total software development effort for process engineering during the last two years). Considering process engineering supported with modelling, one should first have a firm understanding of the current software process (Heineman et al., 1994). Therefore, even a thesis study on facilitating understanding and communication of the software process with process modelling was carried out during the years 1996-1997 in PMR Terminals. The study, to which this report is direct continuation, resulted in the four software process modelling principles described earlier in Section 2. These four principles were successfully applied in describing software process as it is currently enacted and almost all the actors of the software process, about twenty at the time, participated in this modelling exercise. The remainder of this paper focuses on sharing the experience associated to a prescriptive modelling experiment, which was based on the preceding descriptive modelling and guided by the four modelling principles.

Prescribing a desired process

Codifying an existing process through descriptive modelling is seldom enough. By contrast, descriptive modelling usually reveals many shortcomings and bottlenecks in software development, which can effectively be tackled with prescriptive modelling. Furthermore, descriptive models are often rather informal descriptions of the software development including a mixture of explicit and implicit information. At least this was the situation in our case. Therefore descriptive models can't effectively be utilised to support and guide the actual work. These were the main reasons for initiating prescriptive modelling in PMR Terminals: aspiration to improve the quality and performance of the software process and to replace the ambiguous descriptive model with a more formal prescriptive one, which could be supported with process support technology. The same aspiration to improve the quality and performance of software process has influenced the development of the CMM including its organisational process focus and process definition key process areas (KPAs), which are under the spotlight in this paper. Unfortunately, because of the CMM's universal nature, it does not describe how the KPAs should be applied in practice. This lack of guidance has also been recognised by Demirors and Demirors (1998).

Shift From Descriptive to Prescriptive Modelling

The modelling process applied to the prescriptive modelling was similar compared to the one applied in the preceding descriptive modelling including separate phases for data collection, process modelling language engineering, preliminary modelling and process actor participation. Moreover, also the graphical PML used in descriptive modelling was utilised in prescriptive as well. However, the shift from descriptive to prescriptive modelling introduced changes to our modelling approach and PML correspondingly.

In descriptive modelling, when the objective was to facilitate communication and understanding, one of the most essential factors in the modelling was the visuality and understandability of the PML. Therefore, the visual PML, which was based on a traditional DFD (Data Flow Diagram) technique, mixed all substantial process perspectives listed in Curtis et al. (1992) in order to produce rich process diagrams, as was required by the context (see Rossi and Sillander, 1998b). Furthermore, during the descriptive modelling, we abandoned the ambition for exact formality in the name of understandability by concentrating on big informal and visual process diagrams hanged on walls (see a block of a such an informal process diagram left in the Figure 2). This way the process actors participating in the modelling were encouraged to freely comment upon the process model under construction.

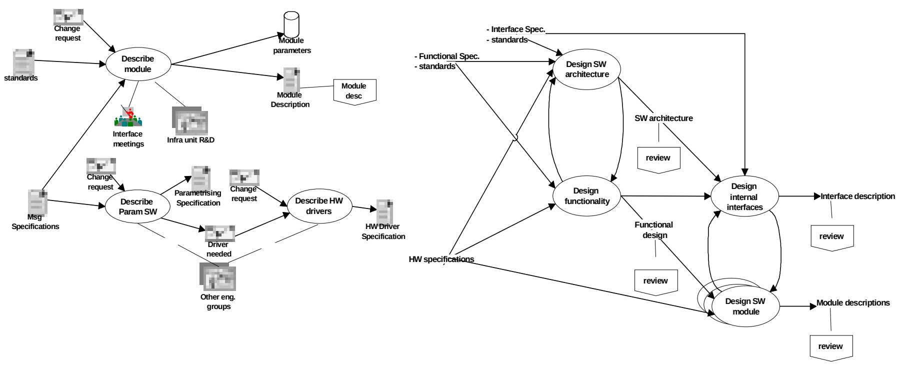


Figure 2. A Block of the SW Design Phase from the descriptive (left diagram) and prescriptive (right diagram) process model.

Prescriptive modelling, on the other hand, requires formality and accuracy while pushing visuality to the background. In our case, the prescriptive modelling took place in two accommodating rounds. During these rounds the ambiguous descriptive model with more or less formal information and details was elaborated to a more formal and explicit prescriptive model. This included continuing the terminology clarification work initiated during the descriptive modelling. Through PML engineering we simplified the conceptual structures and their corresponding representation in the original PML in order to increase its formality and accuracy. In the same vein, we separated the different process perspectives, namely functional, behavioural, organisational and informational (Curtis et al., 1992), which were previously modelled together with one PML. Thus, from the prescriptive modelling point of view, mixing all main process perspectives in one graphical PML yielded cluttered models overloaded with different symbols, which were required in the descriptive modelling. Figure 2 illustrates how the graphical representation evolved during the modelling. Moreover, similarly to the descriptive modelling, the prescriptive modelling was carried out without any special tool support. We merely utilised a computerised drawing tool supplemented with the symbols of our graphical PML.

The principles in prescriptive modelling

In our experiment, the four fundamental software process modelling principles, originally introduced for descriptive modelling, were applied successfully in prescriptive modelling. However, the implications, focus and the mutual relationships of the principles turned out to be somewhat different than in descriptive modelling.

The principle of flexibility. The principle of flexibility was applied throughout the modelling process as in descriptive modelling. The principle was utilised to accommodate the modelling process for prescriptive purposes. This time, however, it was adapted in a more controlled manner meaning, for example, that ad-hoc changes to the PML were no longer allowed, not even in the name of understandability. Instead, the objective behind the application of the principle was the ambition to flexibly increase the formality of the modelling.

The principle of PML engineering. The main tool in applying the principle of flexibility to practice was PML engineering. Thus, we moulded the PML to fit the requirements of prescriptive modelling - mainly formality and accuracy. The PML engineering focused primarily on two interrelated aspects of the PML, conceptual structures and process perspectives. First, based on the feedback from descriptive modelling, we were able to direct the modelling on process aspects considered essential. Thus, the number of different concepts and PML symbols were cut down in order to focus the modelling on crucial process information. Besides improving clarity and accuracy, this also facilitated the documentation of the process in a process manual. Thus, we found it too toilsome to attach large process diagrams overloaded with different symbols into text documents or in any other available electronic media. Moreover, the PML was further supplemented with text and table notation for specifying the details associated to the different phases of the software process, which were left unspecified in the graphical PML. The second aspect, closely related to the first one, affected by the PML engineering was the perspectives included in the modelling (process ontology). In order to decrease the ambiguity caused by multiple perspectives mixed together, the perspectives were separated and the functional one was strengthened over the others. The reason for separating the perspectives and strengthening the functional one was due to the feedback from the first experiments on tailoring the prescriptive model into practice. Thus, the feedback indicated that the activities and data flows between them are most important to be modelled. Focusing more on logical connections of process phases and activities over the temporal relations mainly did the strengthening. We had previously been trying to combine both in one PML. To summarise the perspectives evolution, the graphical part of the PML was engineered to cover the functional perspective while the other perspectives deemed important, informational and organisational, were covered by the new textual part of the PML. The behavioural perspective, on the other hand, was no longer considered worthwhile to model since the temporal relations are ultimately planned and described when tailoring the process model into practice (e.g. in project planning).

The principle of using multiple data sources. During the prescriptive modelling, the principle of using multiple data sources under went perhaps the most significant evolution. When the modelling shifted to prescriptive direction, exhaustive data collection was no longer needed. This was because now the modelling was based on the already existing understanding of the current process and a vision of the desired one achieved through descriptive modelling and internal CMM assessments. Furthermore, the process data collected during the descriptive modelling was on the hand in the prescriptive one as well. However, the principle did not become totally obsolete since software documentation study and participant observation were still used as data sources to maintain the grasp of the current state of software process. Participant observation, in particular, focused on the collection of informal process data through participation in meetings, reviews and everyday discussion. Furthermore, survey of the existing literature was utilised as a new data source. This source was utilised in searching existing solutions related to software development, i.e. best practices to be utilised in our context.

The principle of process actor participation. Finally, the principle of process actor participation was applied successfully in the prescriptive modelling as well. In our experiment, the local process engineer worked as a modelling facilitator carrying the main responsibility of the modelling including the documentation of process diagrams and supplementary text and tables. This responsibility included also the involvement of process actors in modelling. The key process actors participating in modelling included software team leaders, senior designers and project managers covering approximately half of the process actors (about 20 out of 40 at a time), on the other hand, contributed their expertise to the modelling mainly through several design meetings, remote comments and final process model review. This degree of participation may be observed as a neglect when the ideal situation is that process actors define their process themselves. However, being under constant time-pressure, the resource scarce software engineering industry is unlikely to absorb such an intensive process modelling effort. This was exactly the situation in our case. It also became clear that when creating a prescriptive process model it is not that crucial to have all process actors participating. This is because now the modelling is based on the common understanding achieved through full participation in descriptive modelling supplemented with internal CMM assessments.

Discussion

Literature is full of process models for different software development situations and needs. These models include such as traditional life-cycle models (see, e.g., Boehm, 1981) and process descriptions included in method specifications (see, e.g., Yourdon, 1989). Based on these models it could be argued that software process could be modelled without any knowledge of the process context and its current maturity. However, we claim that when a process model is constructed for a given environment, the modelling has to be done in regard to the requirements and characteristics of that specific context. This way it is ensured that the model is realistic and can be instantiated into practice. In PMR Terminals, the discussed prescriptive model has successfully been instantiated into practice and the feedback received from process actors has been encouraging. The feedback has indicated increased understanding of the software process among the process actors, rise in project planning accuracy, better conformity between projects and improved guidance for actual software development. Moreover, the process model has offered a solid foundation for further process improvement, which we strive in small incremental steps tied to the capabilities of the context as well as to the feedback collected from the practice. Thus, because of the intricacies of software development, we believe that it is impossible to create and introduce a model of a desired process tied to the specific context right on the first try. On the contrary, the model needs to be revised and further developed based on the feedback collected from its use in practice. Feedback from testing the model in practice is required in order to bring the modelling into more detailed levels. It may also be necessary to introduce an unfinished model into practice because of too few process engineering resources and the great need for a new model. Thus, it is often necessary to first create a coarse-grained draft of a process model, introduce it into practice and then further develop it based on the direct feedback.

Conclusions

In this paper, we have reported an experiment of prescriptive modelling in an industrial software development context. The objective of this experiment was to explore two research questions "what kind of an effect does the shift from descriptive to prescriptive modelling have on the modelling" and "are the four software process modelling principles, originally introduced for descriptive modelling, applicable to prescriptive modelling". As anticipated, prescriptive modelling imposed amplified requirements for formality and accuracy when compared to descriptive modelling. This included the need to make a more clear distinction between the different perspectives modelled. Thus, we had to separate the representation of the different process perspectives, which in descriptive modelling were combined together at the expense of formality and accuracy. From this we concluded that it is advantageous or even necessary to separate different process perspectives, e.g., with different PMLs in order to build a process model, which can be used to support and guide the actual software development. In our case, we did not introduce any new PMLs for the prescriptive modelling but engineered the one applied in descriptive modelling to suit the needs of prescriptive modelling. However, we assume that when the modelling is iterated as continuous process improvement proposes, the need to introduce a more formal PML or PMLs for depicting the different process perspectives and details more carefully increases. Second, the modelling experiment proved that the four software process modelling principles, originally introduced for descriptive modelling, suit prescriptive modelling as well. Although the implications, focus and the mutual relationships of the principles turned out to be somewhat different compared to those in descriptive modelling. Thus, based on our modelling experience we argue that the four principles are applicable to similar modelling cases, which take place in a comparable process context with similar modelling objectives. However, the principles alone are not enough but they provide guidelines for building a process modelling process and method in accordance with the characteristics of modelling context.

References

Avrilionis, D., Belkhatir, N., Cunin, P. A unified framework for software process enactment and improvement, In Proceedings of the 4th international conference on the software process, IEEE Computer Society Press, 1996.

Bandinelli, S. C., Fuggetta, A., Ghezzi, C. Software process model evolution in the SPADE environment, IEEE Transactions on Software Engineering (19:12), 1993.

Basili, V. R., Rombach, H. D. The TAME project: Towards improvement-oriented software environments, IEEE Transactions on Software Engineering (14:6), 1988.

Boehm, B.W. Software engineering economics, Prentice-Hall, 1981.

de Bunje, T., Engels, G., Groenewegen, L., Matsinger, A., Rijnbeek, M. Industrial maintenance modelled in SOCCA: An experience report, In Proceedings of the 4th international conference on the software process, IEEE Computer Society Press, 1996.

Christie, A., M. A graphical process defintion language and its application to a maintenance project, Information and Software Technology (25:6/7), 1993.

Curtis, B. Kellner, M. I., Over, J. Process modelling, Communications of the ACM (35:9), 1992, pp. 75-90.

Deiters, W., Gruhn, V. The funsoft net approach to software process management, International Journal of Software Engineering and Knowledge Engineering (4:2), 1994, pp. 229-256.

Demirors, O., Demirors, E. Software process improvement in a small organisation: Difficulties and suggestions, In Proceedings of the 6th european workshop on software process technoloqy, Lecture Notes in Computer Science, Springer-Verlag, Berlin Heidelberg, Germany, 1998.

Dutton, J. E. Commonsense approach to process modeling, IEEE Software (July), 1993.

Finkelstein, A., Kramer, J., Nuseibeh, B., eds. Software process modelling and technology, Advanced Software Development Series (ISBN 0-86380-169-2), Research Studies Press Ltd. (John Wiley), 1994.

Heineman, G. T., Botsford, J. E., Caldiera, G., Kaiser, G. E., Kellner, M. I., Madhavji, N. H. Emerging technologies that support a software process life cycle, IBM Systems Journal (33:3), 1994, pp. 501 - 529.

Hirschheim, R., Klein, H. Paradigmatic influences on information systems development methodologies: Evolution and conceptual advances, Advances in Computers (ISBN 0-12-012134-4), 34, 1992, pp. 293-392.

Höltje, D., Madhavji, N. H., Bruckhaus, T., Hong, W. K. Eliciting formal models of software engineering processes, In Proceedings of the 1994 CAS conference (CASCON'94), IBM Canada Ltd. and The National Research Council of Canada, 1994.

Kaiser, G. E., Popovich, S. S., Ben-Shaul, I. Z. A bi-level language for software process modeling, In Proceedings of the 15th international conference on software engineering, IEEE Computer Society Press (May), 1993, pp. 132-143.

Kellner, M. I., Briand, L., Over, J. W. A method for designing, defining, and evolving software processes, In Proceedings of the 4th international conference on the software process, IEEE Computer Society Press, Brighton, 1996.

Kellner, M. I., Hansen, G. A. Software process modeling: A case study, In Proceedings of the 22nd annual Hawaii international conference on systems sciences, IEEE Computer Society Press (2:January), 1989, pp. 175-188.

Klingler, C. D., Schwarting, D. A practical approach to process definition, In Proceedings of the 7th annual software technology conference, 1994.

Kontio, J. Promises: A framework for utilizing process models in process asset management, Unpublished Licentiate thesis at Helsinki University of Technology, 1994.

Koskinen, M., Marttiin, P. Developing a customisable process environment: Lessons learnt and future prospects. In Proceedings of the 6th european workshop on software process technoloqy, Lecture Notes in Computer Science, Springer-Verlag, Berlin Heidelberg, Germany, 1998.

Kumar, K., Welke, R. J. Methodology engineering: A proposal for situation-specific methodology construction. In Challenges and strategies for research in systems development, ed. W. W. Cotterman and J. A. Senn. John Wiley and Sons Ltd, 1992.

Madhavji, N. H. The process cycle, Software Engineering Journal (September), 1991.

Osterweil, L. Software processes are software too. In Proceedings of the 9th international conference on software engineering, IEEE Computer Society Press (2:13), 1987.

Paulk, M. C. Weber, C. V. Curtis, B. Chrissis, M. B. The capability maturity model: guidelines for improving the software process, Addison-Wesley Publishing Company, 1995.

Rossi, S., Sillander, T. A software process modelling quest for fundamental principles, In Proceedings of the european conference on information systems (ECIS), (2), 1998a.

Rossi, S., Sillander, T. A practical approach to software process modelling language engineering. In Proceedings of the 6th european workshop on software process technoloqy, Lecture Notes in Computer Science,Springer-Verlag, 1998b.

Shepard, T., Wortley, C., Sibbald, S. A visual software process language, Communications of the ACM (35:4), 1992, pp. 37-44.

Yourdon, E. Modern structured analysis, Yourdon Press, 1989.

SESSION 9 :

Methods and Tools in SPI

A Software Process Simulator for Software Product and Process Improvement

Paolo Donzelli and Giuseppe Iazeolla
Laboratory for Computer Science
and CERTIA Research Center*
University of Rome "TorVergata"
Roma, Italy
{donzelli,iazeolla}@info.uniroma2.it

Abstract

Software process improvement can benefit from the application of software process quality models with dynamic estimation capabilities. In order to guarantee the dynamic estimation capability, and to deal with the complexity of the modelling problem, this paper proposes the combination of three conventional modelling methods (analytical, continuous and discrete-event) into a unique hybrid multi-level new model, called Dynamic Capability Model (DCM).

DCM is applied to a waterfall-based software process to study the effects of three different quality assurance management policies on given process quality attributes, as effort, delivery time, productivity, rework percentage, and product quality. DCM simulation results can stimulate debate, and provide both qualitative and quantitative suggestions on the ways to change the software process to improve its quality, or the ways to achieve specific organisation's needs.

---

* Work partially supported by the CERTIA Research Center, and by the UNIROMA2-UMD-WVU Co-operation agreement on Concurrent Engineering and Simulation Modelling in Software Process Optimisation.

1. Introduction

Software process quality is a multi-attribute index, spanning from process effectiveness, to process productivity, rework percentage, etc., including product quality itself (defect density, reusability, etc). Process improvement requires managing its complexity by means of appropriate models with "dynamic estimation" capabilities. In other words, the capability to analyse and understand the as-is process, to design the to-be process, to forecast process trajectories for a better project control, to simulate outcomes under different what-if conditions, without affecting the actual environment.

Quality models conventionally used by the software community are of analytical average-type and do not generally hold such capability. Examples of such models are the Function point model [1], the COCOMO model [2], the Rayleigh model [3], and the Reliability Growth model [4]. Besides lacking dynamic estimation, and only providing "average" estimates on the process quality, these models cover a strict subset of the quality attributes, separate the effects of such attributes (e.g. the development effort from the defect density), and do not permit to analyse the process behaviour in a perturbed environment (e.g. changes in product requirements, staff reductions, etc).

Giving a process model the dynamic capability generally requires the introduction of so complex relationships between internal and external variables, that mathematical model solution techniques have to give way to simulation solution approaches. However, existing process-simulation techniques also suffer by drawbacks, in that they only enhance some process aspects to detriment of the others. This is since such techniques are either of discrete-type (discrete-event queuing based) [5], or of continuous-type (system dynamics based) [6] [7], and only rarely give a combination thereof.

It is the paper view that to give a model the dynamic capability property one has to combine all three above-mentioned modelling methods (the average-analytical modelling method, the discrete-type and the continuous-type one) into a hybrid modelling method. In this view, the paper introduces a predictive hybrid model, called the Dynamic Capability Model (DCM). The hybrid method is applied in DCM according to a two-level abstraction framework. At the higher abstraction level, the discrete-event method is used, while the analytical and the continuous-type methods are used at the lower abstraction level. A simulation technique is used to solve the hybrid model. The QNAP2 simulation package [8] is used, which includes many features that support hybrid modelling.

It is generally argued that simulation solutions are unlikely able to give exact forecast of the real process behaviour. However, it is this paper view that they nevertheless give projections on how the process would behave under given assumptions on external and internal factors. They stimulate debate and provide a way to learn about how to improve process quality. To sustain this view, some application examples are presented, which predict and analyse the effects of process-management factors (reviews and testing effectiveness) on process quality attributes as effort, delivery time, productivity, rework percentage and product quality.

This work is one of the results of the joint co-operation between the University of Roma “Tor Vergata”, the Enterprise-University Consortium CERTIA, the Software Engineering Laboratory of the University of Maryland and the CERC Research Center of the University of West Virginia, on the "Concurrent Engineering and Simulation Modelling in Software Process Optimisation" enterprise-university project.

The paper is organised as follows. Section 2 gives a brief overview of DCM, and Section 3 describes an example use of DCM to study process quality.

2. DCM for the waterfall paradigm

A software process based on the waterfall paradigm is taken into consideration in this paper, and modelled in DCM. According to such paradigm, the software process (illustrated in Figure 1) consists of a series of sequential phases, and the software product is the conclusive artifact of a series of intermediate artifacts, named *requirements*, *specification*, *high-level design*, *low-level design*, *code*, *system-tested code* and *acceptance-tested code*. Such artifacts are also referred to as primary artifacts.

Although phases are sequential, their respective activities can run concurrently, because of the simultaneous execution of work activities (that generate primary artifacts) and rework activities (necessary to fix defects or to introduce requirement modifications). Artifacts generated by the rework activities are referred to as secondary artifacts. They are named *defect reports* or *correction reports* if generated by activities aiming at fixing defects. They are instead named *changes* or *increments* if generated by activities that introduce modifications due to requirements instability. The waterfall process thus consists partly of sequential and partly of concurrent activities.

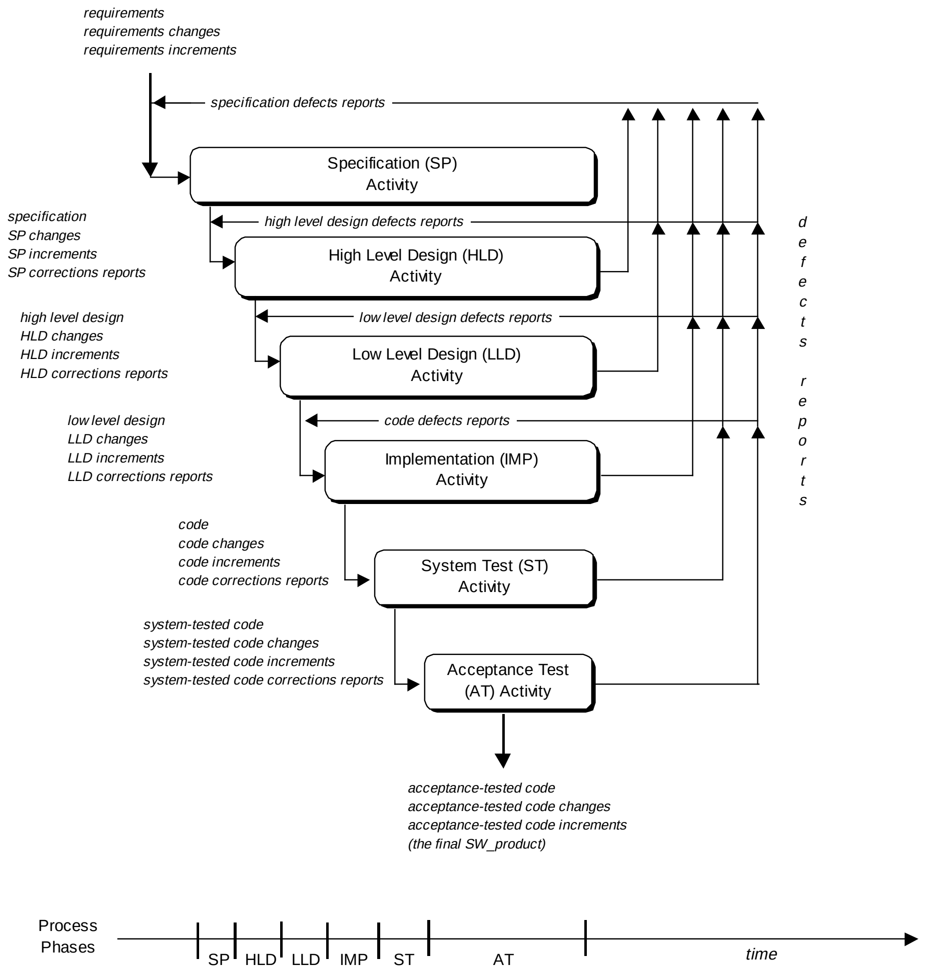


Figure 1- The modelled software process

Activities are distinguished into development activities, and testing activities. In Figure 1, the development activities are the Specification (SP), the High Level Design (HLD), the Low Level Design (LLD), and the Implementation (IMP) activity. The testing activities are the System Test (ST), and the Acceptance Test (AT). The (primary or secondary type) artifacts various activities yield are reported on the arrowhead sides in Figure 1.

The Figure 1 process is translated in DCM according to a two-level abstraction framework: the higher and the lower abstraction level, described in Section 2.1 and 2.2, respectively.

The process quality attributes taken into account by DCM are effort (E), delivery time (T), productivity (P), rework percentage (RWK), product defect density (DFD) and many sub-attributes thereof (process staffing profile, staffing profile over single activities, duration of each phase, final product size, etc.). However, for the sake of conciseness, in this paper we concentrate on the study of only a few of them.

2.1 The DCM higher abstraction level

At the higher abstraction level, the discrete-event modelling method is used: the process is modelled by a discrete-event queueing network. The queueing model is a direct replica of the software process. Service stations are used to represent activities and sub-activities, whilst circulating customers are used to represent artifacts that move from one activity to another and that are enqueued on the entrance of a given activity and wait for services.

Figure 2 illustrates the queueing network used to model the HLD activity. The main service stations are the "work station", the "external rework station", the "internal rework station" and the "review station".

The “work station” simulates the development of the *high-level design* artifact on the basis of the demand submitted in input by the *specification* artifact.

Basing on the demand in input for *SP changes*, or *SP increments*, the “external rework station” simulates the modification of the already released *high-level* *design* artifact, and yields the corresponding output artifacts (*HLD changes* and *HLD increments*). Similarly, basing on the demand in input for *SP corrections reports* or *HLD defects reports*, the “internal rework station” simulates the correction of the released *high-level design* artifact, and yields the corresponding *HLD corrections reports*.

Finally, the “review station” simulates the review performed on the *high-level design*, the *HLD changes*, and the *HLD increments* artifacts. No review is performed on *HLD correction reports*, assumed with no defects. In other words, it is assumed that the correction activities (simulated by the “internal rework station”) inject no defects.

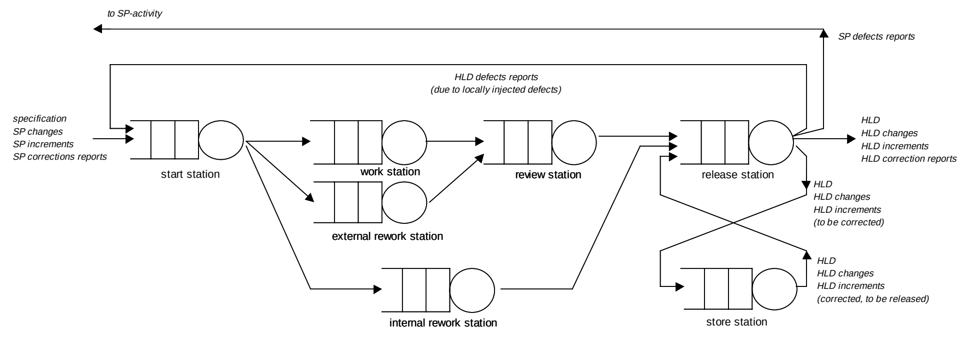


Figure 2 - Higher abstraction level of the HLD activity

The “start”, “release” and “store” stations in Figure 2 are assumed to be zero service-time stations, since they perform just co-ordination activities. In particular: the “start station” directs the input artifact to the appropriate service station, depending on its type, whereas the “release station” and the “store station” take care of the releasing of the artifacts.

The *high-level design, the HLD changes* and *the HLD increments* are released by the “release station” only if no defects have been found by the “review station”. If, however, some defects have been found, the “release station” creates the corresponding *defects reports* (e.g. *HLD and SP defects reports*) and sends them back to the activities responsible for the defects. The faulty artifacts are then sent to the “store station”, where they are held until all the *corrections reports* corresponding to the released *defects reports* are received.

2.2 The DCM lower abstraction level

The lower abstraction level gives the implementation details of the service stations (or sub-activities) introduced at the higher abstraction level. The analytical and the continuous modelling methods are used at this level. In particular, each sub-activity is modelled either by an analytical average-type function, or by a continuous type time-varying function (or by a combination thereof). Such functions are used to express the amount of resources (e.g. personnel), or time, or effort (person-week) that service stations use to simulate the corresponding sub-activities.

Figure 3 shows the implementation details of the "work station", one of the main service stations depicted in Figure 2, and of its corresponding input and output artifacts. The station simulates the development of the *high-level design* artifact, starting from the *specification* artifact.

The *specification* and *high-level design* artifacts are described by a set of four attributes: name, size, development effort and defectiveness. Attributes name and size are of immediate evidence. The attribute defectiveness is described by an array whose j-th element is the amount of defects injected into the artifact by the j-th development activity (j = SP, HLD, LLD, IMP). The attribute total development effort (W1 for the *specification* and W1+W for the *high-level design*) is the total effort that has been spent to develop the artifact itself since the beginning of the process. Thus, it encompasses also the effort spent to develop all the artifacts from which it has been derived.

The values of the attributes of the *high-level design* artifact, together with the amount of time, T, ("work station" service time), and of personnel over time, E(t), required to develop such an artifact, are derived as illustrated in Figure 3. All of these quantities may have random deviations, and are therefore simulated according to gaussian-like probability distributions. More in detail, the average size of the *high-level design* artifact is first derived from the size of the *specification* artifact by use of the COCOMO-like size estimator block. The corresponding random size is then obtained by use of the gaussian-like pseudo-random generator. This value is then given to the COCOMO-like time estimator block, to obtain the random service time (T), and to the COCOMO-like effort estimator block to obtain the random development effort (W). That is, as shown by the shaded area in Figure 3, the effort simulated by the "work station" to develop the *high-level design* starting from the *specification*.

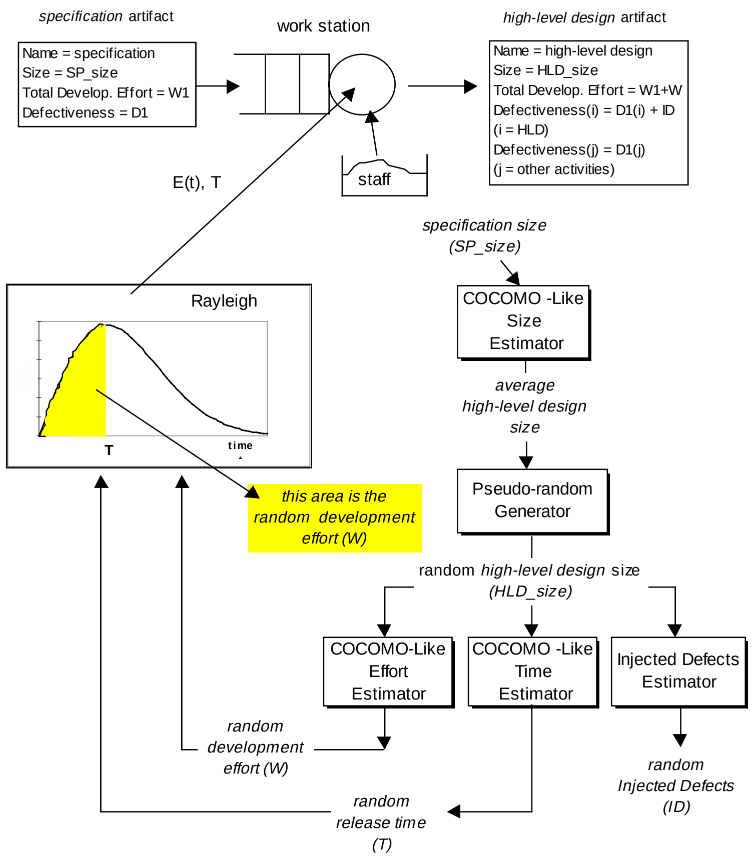

specification artifact
Name = specification
Size = SP_size
Total Develop. Effort = W1
Defectiveness = D1
work station
staff
high-level design artifact
Name = high-level design
Size = HLD_size
Total Develop. Effort = W1+W
Defectiveness(i) = D1(i) + ID
(i = HLD)
Defectiveness(j) = D1(j)
(j = other activities)
E(t), T
specification size (SP_size)
COCOMO -Like Size Estimator
average high-level design size
Rayleigh
T
time
this area is the random development effort (W)
Pseudo-random Generator
random high-level design size (HLD_size)
COCOMO-Like Effort Estimator
COCOMO -Like Time Estimator
Injected Defects Estimator
random development effort (W)
random Injected Defects (ID)
random release time (T)

Figure 3 - Lower abstraction level of the "work station" in the HLD activity

The random development effort (W) is then added to the value of the total development effort attribute of the *specification* artifact (W1) to obtain the value of the corresponding attribute of *high-level design* (W1+W).

On the basis of T and W, the amount of required personnel, E(t), is finally obtained using the Rayleigh function [3]. According to Putnam's assumption [3], the *high-level design* artifact is released when E(t) reaches its peak. Moreover, unlimited staff availability is assumed. In other words, it is assumed that the staff pool in Figure 3 can always supply the personnel necessary to fit the E(t) curve demand for personnel. DCM, however, can easily accept more realistic assumptions on finite staff pools.

The amount of defects injected into the *high-level design* artifact (injected defects, ID) is obtained from the injected defect estimator block, as a by-multiplication of the random size of the *high-level design* and the expected defect density (defects per unit of size). Defect density is a parameter used in DCM to summarise the effects of various factors (personnel skill, team structure, supporting tools, programming language, product type, etc) on the defectiveness of a given development activity. DCM, however, can easily accept more elaborate defect injection models, as for example models in [9].

The derived ID is then summed to D1 (*specification* defectiveness) to obtain the *high-level design* defectiveness.

More details on the analytical derivations of the functions used to model this station (and all the other stations in Figure 2) can be found in [10], [11] and [12].

3. Example use of DCM to study process quality

The DCM simulator can be used to predict and analyse the effects of various process factors on the process quality (e.g. effort, delivery time, and productivity). Even if, in many cases, such effects are known facts at qualitative level, they are made quantitative by use of the DCM simulator. In this Section, the quantitative study of the so-called “find as much as early as possible” defect detection strategy is made. In qualitative terms, it is already known that such strategy improves the process schedule, effort and the final product quality. The use of DCM will provide a quantitative evaluation of such advantages.

In such a perspective, DCM will be used to study the effects of three different defect detection policies (P1, P2 and P3) in a software development scenario with stable requirements. Stable requirements meaning that the size of the initial requirements (assumed to be of 1500 Function Points) does not change during product development.

The three policies are characterised by different allocations of the defect detection resources along the life cycle, however yielding the same final product quality (simply measured in DFD). In DCM this can be expressed by assuming different defect detection effectiveness (DDE) for the process defect detection activities (SP-review, HLD-review, LLD-review, IMP-review, ST and AT). In fact, DDE is a DCM input variable that can be used to specify the detection effectiveness of a review or testing sub-activity in terms of percentage of removed defects.

The values of DDE adopted for P1, P2 and P3 are reported in Table 1. In summary, it is assumed that in P1 (or Early Detection policy) the DDEs are higher in the initial activities of the lifecycle, in P2 (or Middle Detection policy) the DDEs are higher in the middle activities of the lifecycle, in P3 (or Late Detection policy), the DDEs are higher in the final activities of the lifecycle.

Table 1-Defect detection effectiveness for Early, Middle and Late policies

Policy	SP review DDE	HLD-review DDE	LLD-review DDE	IMP-review DDE	ST DDE	AT DDE
Early Detection	95%	95%	95%	75%	50%	50%
Middle Detection	10%	40%	80%	80%	55%	50%
Late Detection	5%	5%	5%	20%	60%	95%

Comparison is made by use of the DCM simulator, analysing how the values of the attributes E, T, P, and RWK (DFD is constant) change from P1 to P2 to P3.

Figures 4 and 5 illustrate the simulation results for the personnel over time (E(t)) in the Early and Late Detection policies. They show that when the Early Detection policy is applied a reduction of effort (E), represented in Figures 4 and 5 by the shaded area, and of delivery time (T) is obtained. In particular, the effort moves from 581 to 519 person-week, whereas the delivery time moves from 102 to 82 weeks.

Furthermore, it can be observed that in the Early Detection policy case, more time and effort are spent during the development phases (SP, HLD, LLD, and IMP phases) rather than during the testing ones (ST and AT phases). On the contrary, when the Late Detection policy is applied, more time and effort is spent during the testing phases rather than during the development ones.

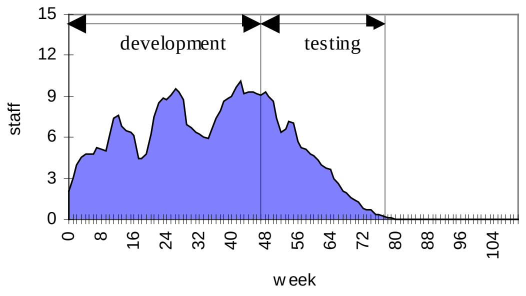


Figure 4 – Personnel over time for the Early Detection policy

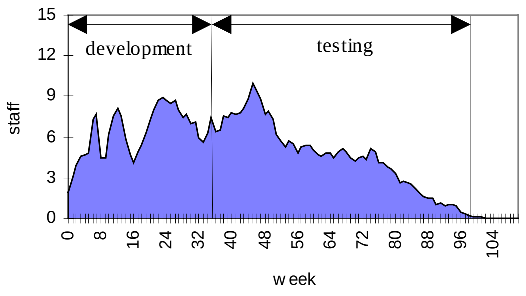


Figure 5 – Personnel over time for the Late Detection policy

To validate such simulation results, further simulation experiments have been carried out to explain the process behaviour. By use of DCM Figure 6 has been first obtained. This picture shows the defect detection patterns for the defects injected by the HLD activity (for the Early and the Late Detection policies). The histogram bars along one column (i.e. Late or Early policy) give the number of defects that have been injected by the HLD activity and have been detected by the subsequent defect detection activities (HLD-review, LLD-review, System Test and Acceptance Test). For example, the IMP-review bar in the "Late" column indicates that about 50 defects, injected by the HLD activity, have been found during the IMP review in the Late Detection policy case. Thus, the heights of the histogram bars along one column sum up to the total amount of defects which have been injected during the HLD activity and which have been detected by the subsequent defect removal activities (e.g. 340 in the Figure 6 case).

In summary, Figure 6 shows that in the Early Detection policy most of the defects are discovered and reworked locally, i.e. during the same activity that injected them, whereas, in the Late Detection policy, most of the corrections are not performed locally, but during the testing phases and this contributes to move time and effort towards the testing phases (as in Figures 4 and 5).

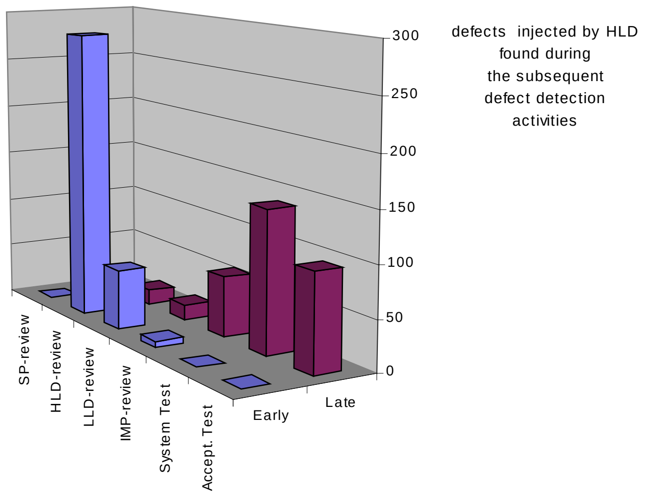


Figure 6 - Defect detection patterns for Early and Late Detection policy

In addition, although the total number of removed defects is the same (340 in Figure 6), in the Late Detection policy case they are discovered later, leading to a higher consumption of resources during the defect correction cycles. In terms of process quality, this contributes to the higher effort, delivery time, rework percentage (from17% to 25%) on one side, and to the lower productivity (of the 13%), on the other, for the Late Detection policy case in comparison with the Early one. A further validation of the observed differences among the three policies can be found by considering the total amount of effort spent in removing defects. In particular, to this purpose, further simulation experiments have been carried out, which show that, moving from the Early to the Late Detection policy, the total effort spent by review and testing activities increases by about the 18%.

To further illustrate the utility of DCM, Figure 7 has been obtained, which gives a synthetic view of the normalised values of the process quality attributes for all the 3 policies (P1, P2, and P3). It can be seen that moving from the Early to the Late Detection policy, the effort (E), delivery time (T) and rework percentage (RWK) increase, whereas the productivity (P) decreases. As assumed, the final product quality (DFD) at the lifecycle end is the same for the three policies.

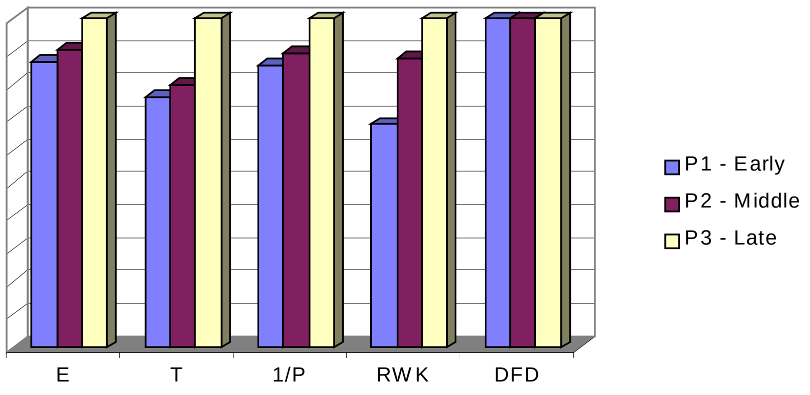


Figure 7 - Comparison of quality attributes for Early, Middle and Late policies

4. Conclusions

Understanding and managing the quality of the software process is the goal of many organisations. On the other hand, the modelling process is quite complex. In order to deal with such a complexity, a combination of three methods (analytical, continuous and discrete-event) into a unique hybrid multi-level modelling methodology with dynamic capabilities is proposed.

The methodology is applied to a waterfall-based software process to produce a Dynamic Capability Model, called DCM. DCM is then used to predict the effect of process-management factors (e.g. reviews and testing effectiveness) on some process quality attributes, as effort, delivery time, productivity, and product quality. Applications of the model show that simulation results can provide both qualitative and quantitative suggestions about how to change the software process to improve its quality or to fulfil the organisation's goals.

Plan for future work includes the extension of the model to less conventional process paradigms, such as the spiral paradigm and the concurrent engineering paradigm.

References

[1] Albrecht A.J. "Measuring application development productivity", Proceedings of IBM Application development Joint SHARE/GUIDE Symposium, Monterey, CA, 1979.

[2] Bohem B.W. Software Engineering Economics. Prentice-Hall, N.J., 1981.

[3] Putnam, L.H. and W. Meyer. Measures for Excellence: Reliable Software on Time within Budget. Prentice-Hall, N.J., 1992.

[4] Fenton, N.E., and Pfleeger S.H. Software Metrics: A Rigorous and Practical Approach. International Thomson Computer Press, UK, 1997.

[5] Hansen, G.A. "Simulating Software Development Processes", Computer, pp 73-77, IEEE, January 1996.

[6] Abdel-Hamid, T.K.. "System Dynamics Models", Encyclopaedia of Software Engineering, pp 1306-1312, Wiley&Sons Inc., NY, 1994.

[7] Calavaro, G.F., Basili V.R., Iazeolla G. "Simulation Modeling of Software Development Process", Proceedings of the 7th European Simulation Symposium, Erlangen-Nuremberg, GE, October 1995.

[8] SIMULOG. QNAP 2 User Guide ver. 9.3. Simulog, 1986.

[9] Stutzke M., Agrawal M., Smidts C. "A stochastic model of human error during software development", Proceedings of the combined 9th European Software Control and Metrics Conference and the 5th conference for the European Network of Clubs for Reliability and Safety of Software, pp 302-310, Roma, Italy, May 27-29, 1998.

[10] Donzelli, P. and Iazeolla G. "Performance Modeling of Software Development Processes", Proceedings of 8th European Simulation Symposium Simulation, pp 339-346, Genova, Italy, October 1996.

[11] Donzelli, P. Performance Modelling of the Software development Process, Ph.D. Thesis, University of Rome "Tor Vergata", Roma, Italy, 1997.

[12] Donzelli, P. and Iazeolla G. "A multi-level hybrid approach to model software development processes", Proceedings of 9th European Simulation Symposium Simulation, Passau, Germany, October 1997.

Repurposing Requirements: Improving Collaborative Sense-Making over the Lifecycle

Albert M. Selvin
Bell Atlantic Corporation
400 Westchester Avenue
White Plains, NY 10604 U.S.A.
Email: albert.m.selvin@bellatlantic.com

Simon J. Buckingham Shum
Knowledge Media Institute
The Open University
Milton Keynes, MK7 6AA U.K.
WWW: http://kmi.open.ac.uk/sbs
Email: sbs@acm.org

Abstract

This paper suggests *collaborative sense-making* as a way to view the process toward creating mutually intelligible representations of requirements that can serve as bridges between different stakeholder communities over the software development lifecycle. In order to do this, we describe the types of obstacles that can impede *representational literacy* across communities of practice coming together in a design effort. We then offer *representational morphing* as a strategy for addressing these obstacles, and show how it has been implemented in an approach and hypermedia groupware environment in industry use named *Project Compendium*. We conclude by reflecting on the key features of the approach and collaborative tool support which point toward future development in support of representational morphing.

Introduction: mutually intelligible representations

A central concern in software process improvement is to understand, and represent, the perspectives of the different stakeholders in the requirements management process over a product development lifecycle. Requirements management encompasses all treatments of product requirements, from early elicitation of product goals, through detailed requirements analysis, through prioritisation of development efforts over the course of maintenance and enhancement releases. In each of these phases, diagrammatic and textual representations of requirements must be developed, discussed, reviewed and approved by various stakeholders. To ensure that developed products meet customer requirements, representational schemes for requirements must be accessible and useful to designers, programmers, and different user communities. Designing *mutually intelligible representations* which can meet the needs of such diverse groups is a key challenge. Robinson and Bannon (1991) have analysed why and how the meaning of representations will invariably 'drift' as they were passed between different design communities, whilst many others have sought to develop representations which bridge between end-users' and designers' perspectives (e.g. Blomberg and Henderson, 1990; Chin and Rosson, 1998; Muller, 1991).

This paper suggests *collaborative sense-making* as a way to view the process toward creating mutually intelligible representations. In order to do this, we describe the types of obstacles that can impede communication across communities of practice coming together in a design effort. We then offer *representational morphing* as a strategy for addressing these obstacles, and show how it has been implemented in an approach named *Project Compendium*. We conclude by reflecting on the key features of the approach and collaborative tool support which point toward future development in support of representational morphing.

Obstacles to mutually intelligible requirements representations

Requirements management is an inherently divergent process, where different communities have different interests. As they move toward an “anticipated state in the future,” each community complicates the overall design discourse by adding their own discourse of communicative styles, concerns, assumptions, and relationships (Isenmann and Reuter (1997)). Each group’s processes of participation and reification -- representation of requirements and other information in text and diagrams -- must be combined with the others involved.

Members of different communities are always bound by the contingencies of their present situation, in ways they are only partially sensible of. These bindings affect their senses of what is possible, what is good, what is harmful, and what is unworkable. Similarly, each community has its own desires of transcending those contingencies by creating some future state, or some portion of them. Different communities are aware of different contingencies and the (possibly negative) consequences of addressing them, as well as they are aware of potential transcendences and the benefits they imagine will accrue from their realisation.

The challenge we address in this paper is to try and facilitate integration between representations designed by different communities in the face of the inherent obstacles to such integration. We discuss two obstacles next: community-specific literacies and decentered communication.

Obstacle: Community-specific literacies

The different communities in system design have their own professional literacy practices which naturally create communication barriers to other communities. Developers, particularly of large scale systems, use a variety of representations to formally specify behavioural properties (e.g. using mathematical/logical notations; fault tree diagnoses), track information flow and dependencies (e.g. Entity-Relationship and Data Flow Diagrams, State Transition Networks), and so forth. Developers of small to medium systems are unlikely to use as many formal representations of system behaviour as those, for instance, in a large, safety critical system, but their perspective is nonetheless that of implementors, working with abstracted, information-centric models of the work practices to be supported. In contrast, a domain expert/end-user needs representations with obvious correspondences to documents, processes, tools, people, times and places in their work in order to understand the implications of the future system.

Obstacle: Decentered communication

Development of, and discussion about, representations usually occurs under a set of conditions that affect both individual and group communication processes. At least in industry settings, collaborative design sessions often occur in group settings, such as formal or informal meetings, under time and deadline pressure, with semi-understood external constraints (regulatory issues, management imperatives, and the like), internal tensions, and power struggles impinging on the ostensible subject at hand. In many cases, the rush to make decisions and develop solutions and designs—products—means that little attention is paid to developing a shared understanding of the problem space and constructing consensual definitions (Weick, 1993; Weick, 1995). Many participants have limited understanding (and even sheer incomprehension) of portions of the problem space, such as the subject matter, technical issues, political pressures, or factors in the external environment. There is typically more than the printed agenda riding on the outcome, and even the process, of such meetings.

Added to these pressures is the emotional dimension. People in organisations have feelings and emotions about each other (as individuals and as members of "other" groups) as well as about the issues they're discussing. In addition, some people are good at articulating ideas and comments about representations in such meetings; others are not, and thus unable to contribute effectively at the time representations are created, modified, or discussed. All these factors contribute to a decentering of people's communicative competencies, which create obstacles to the development of mutually intelligible representations that will enable development and realisation of the desired future state.

These obstacles can be understood as the (often unspoken) questions that people ask, or that they are troubled by, during the course of representation development. These include:

- "Why are we doing this?"
- "What am I here for?"
- "Why do they insist on (calling W what I know as X/saying that we need Y/ignoring our requests for Z)
- "How is this helping me achieve my (group's) goals?
- "Why do we have to do this in this manner?"
- "How is what we're doing contributing the project's overall progress?"
- "Why aren't my concerns getting put up on the board?"
- "What do these terms mean?"

Understanding the obstacles in terms of collaborative sense-making

Developing and applying requirements representations, whether they are mutually intelligible or not, always happens in a context of shifting and multiple *sense-making* (SM) efforts (Dervin, 1983). Everyone involved is engaged in their own SM effort. There are not only gaps in the languages, frames of reference, and belief systems that people in the different communities of practice have, but gaps between their respective SM efforts—their problematics in the representational situation are different. In many cases, different communities have mutually unintelligible SM efforts, leading to mutually unintelligible representational efforts.

Weick (Weick, 1993) calls for "sensemaking support systems" that can aid the process of constructing "moderately consensual definitions that cohere long enough for people to be able to infer some idea of what they have, what they want, why they can't get it, and why it may not be worth getting in the first place." Dervin's (1983) model of sense-making posits that a persons, or groups, are always attempting to reach a goal, or set of goals. The goals themselves shift in time and place. Some are tacit, some are explicit; some are conscious, some are unquestioningly assumed or inherited. Actors in a situation will continue trying to reach the goal until they are impeded by some obstacle. This obstacle stops their progress and stymies their efforts to continue. In order to resume their progress, they need to design a movement around, through, over, or away from the obstacle. The actions they take at the moment of confronting the obstacle are *sense-making actions*, which can be understood as attempting to answer a set of questions: What's stopping me? What can I do about it? Where can I look for assistance in choosing/taking an action?

In systems development contexts with multiple groups involved (multiple stakeholders, as well as multiple individuals within the various groups) the problem is compounded, because all the groups and individuals involved are engaged in their own (overlapping/conflicting) sense-making efforts. Each group is always attempting to reach its goal. The goals themselves are sometimes shared, sometimes divergent. Group 1 believes that Group 2 is happily trying to achieve Goal 1. In reality Group 2 is trying to achieve Goal 2. As time goes on, obstacles interfere with each group's progress. The obstacles appear for the different groups at different times, in different forms, at different levels of comprehension and articulation. Each group attempts to find ways through, around, over, or away from their own obstacles. They communicate with each other and with members of other groups; this either helps them along or hurts their efforts. Having done this, the groups, assuming they have been able to gather good information, communicate effectively, and make decisions, continue on their way, making progress towards their goal(s). However, the goals are still not well understood by the different groups, and it is unlikely they have developed any degree of shared understanding.

Representational morphing as a collaborative sense-making approach

The act of representing requirements *always* occurs in a contested, shifting, terrain of multiple sense-making efforts. Consequently, tools are needed to help each community see and appreciate each other’s goals, obstacles, and strategies; to learn from each other, as opposed to a simplistic model that all that is required is to retrieve, represent and integrate domain knowledge. Tools can help each community to understand elements of other communities’ literacy strategies, and incorporate them into their own.

We propose *representational morphing* as an approach to the design of such tools. Representational morphing is the ability to transform a representation, or elements of a representation, at any moment, with little or no delay, in order to respond to the sense-making requirements of one or more of the communities involved in a design effort. By doing so, the different group can read and/or write the representation according to their own literacies. Representational morphing further supports the ability to incorporate these new writings and transform them into other forms or contexts to aid in the sense-making efforts of the other involved communities.

The following section introduces an approach -- Project Compendium -- developed with the concerns of representational literacy and collaborative sense-making in mind, and describes how the approach’s developers have supported representational morphing to date. It should be emphasised that the description that follows are early steps toward full support and are offered as an illustration of the possibilities, rather than as a complete solution.

Description of the Project Compendium environment

Project Compendium (Selvin, 1998; Selvin, 1999) is a system that knits together off-the-shelf tools and documents to create a customised environment for collaborative project work**.** The system provides the capability to convert the elements of various types of common documents (for example, email, word processing, and spreadsheet documents) into nodes and links in a hypertext concept-mapping tool. Once in the tool, users can create associations between those elements while preserving the original set of associations in the source document, and export the associated elements in new documents. Project Compendium also prescribes a set of techniques that can be used within the off-the-shelf tools themselves to add keyword coding and labelling schemas that allow the cross-referencing of ideas and elements from the different sources within the hypertext tool. The approach also provides a set of representational forms that allow groups to do collaborative modelling using hypertext representations.

The system supports a wide range of project activities, including issue-tracking, modelling, planning, analysis, design, and other project management tasks. The system supports facilitated group planning and modelling sessions, conducted either face-to-face or over a network. To use the system, groups define the types or categories of information they are interested in and then create a number of "templates" embodying these categories that can be used by the various tools involved. Documents loaded into or created within the system are represented as collections of nodes and links corresponding to the relationships between individual ideas (for example, each paragraph in an email is a separate node, linked to a node representing the subject of the email). Once available to the hypertext database in this manner, the individual nodes can be linked to any other nodes in the database. For example, individual points in a meeting minutes document can become "action item" nodes that then reappear in lists of assigned action items to project members, elements of design models, items in a test plan, and so forth.

The approach has been used in more than twenty software development, business process redesign, and other projects at Bell Atlantic, as well as with community organisations, software companies, and research groups to support a variety of collaborative analysis efforts. Project involvements have ranged from small software development teams of two to five people to large process redesign efforts involving dozens of participants in both large group meetings and small sub-group work sessions. Project durations have ranged from several weeks to more than two years (one current software development project team has employed Project Compendium continuously since its onset in 1996).

Database size range from hundreds to more than ten thousand nodes, many of which reappear in many contexts in the database.[27]

One of the central aspects of Project Compendium is the ability for users to define the types or categories of information they are interested in and then create a number of “templates” embodying these categories. Templates follow a question-and-answer format. Questions are drawn from the categories, or attributes of categories, of interest, while the expected answers conform to the rules established for that category or attribute, The format has general similarities to that described by Potts et al (1994)[28], although use of Project Compendium extends beyond the domain of requirements. Figure 1 shows the general form of Project Compendium templates.

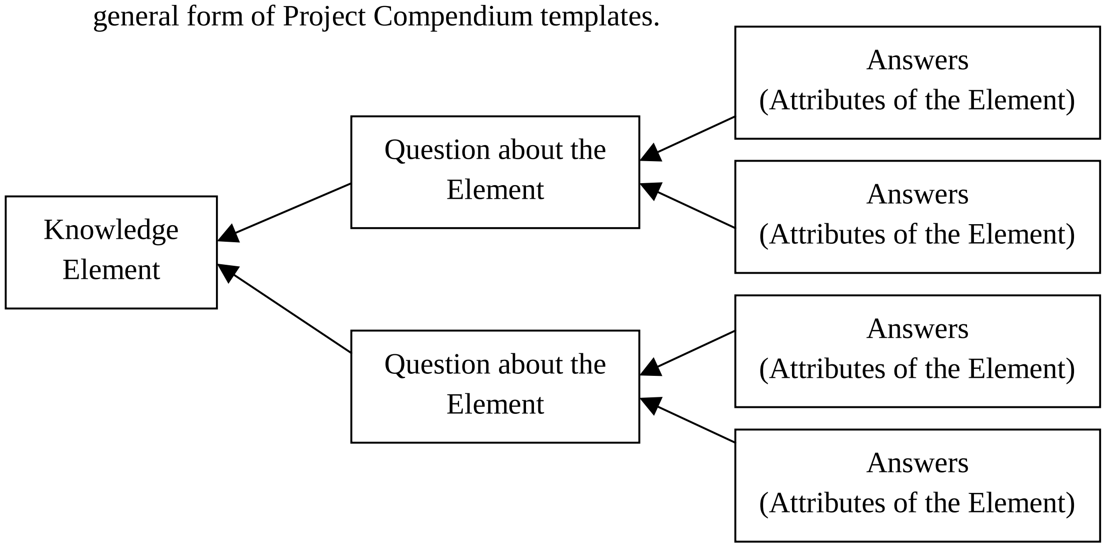

Knowledge Element
Question about the Element
Question about the Element
Answers (Attributes of the Element)
Answers (Attributes of the Element)
Answers (Attributes of the Element)
Answers (Attributes of the Element)

Figure 1. General form of Project Compendium templates

[27] Detailed quantitative analysis of Compendium’s use remains to be conducted. However, to give an indication of scale, from December, 1996 – January, 1999, the "FMT" project team's database has 11,833 nodes in 582 views. 13 users are registered within the system, but this does not reflect the number of participants in meetings facilitated and recorded by the tool, who would number approximately 40-50.

[28] Potts, C., Takahashi, K., Anton, A. “Inquiry-Based Requirements Analysis,” in *IEEE Software*, March 1994.

Some Project Compendium questions and templates are derived from structured modelling approaches, while others grow out of immediate and/or informal concerns of the participating groups. For example, in one project which used Project Compendium, questions about objects and their attributes were inspired by formal object-oriented analysis (Coad and Yourdon, 1991; Jacobson, 1992). Questions about organisations and roles were originally based on elements of the CommonKADS Organisation Model (de Hoog, Kruizinga and van der Spek, 1993). Questions about problems and opportunities in the domain, however, were generated spontaneously by members of the participating groups themselves, in response to domain-specific issues.

Representational morphing in Project Compendium

Project Compendium's users have employed a variety of representational morphing strategies to facilitate collaborative sense-making amongst participating communities and their members. Two will be discussed here: rapid recombination of knowledge elements for ad hoc, opportunistic collaborative representational activities, and early use of a new technique: transformation of template-based concept maps into developer-oriented representations such as data flow diagrams (DFDs) and review documents.

Rapid recombination of knowledge elements for ad hoc representational activities

The first example shows a Project Compendium representational morphing technique that has been employed by many of the approach's users in diverse projects. The example below was taken from a software and business process redesign project. The knowledge elements depicted were originally developed in late 1996-early 1997 as part of the project's early requirements analysis. The map below shows a high-level overview, or collection of maps, containing the requirements specifications for the various modules of the system (Figure 2).

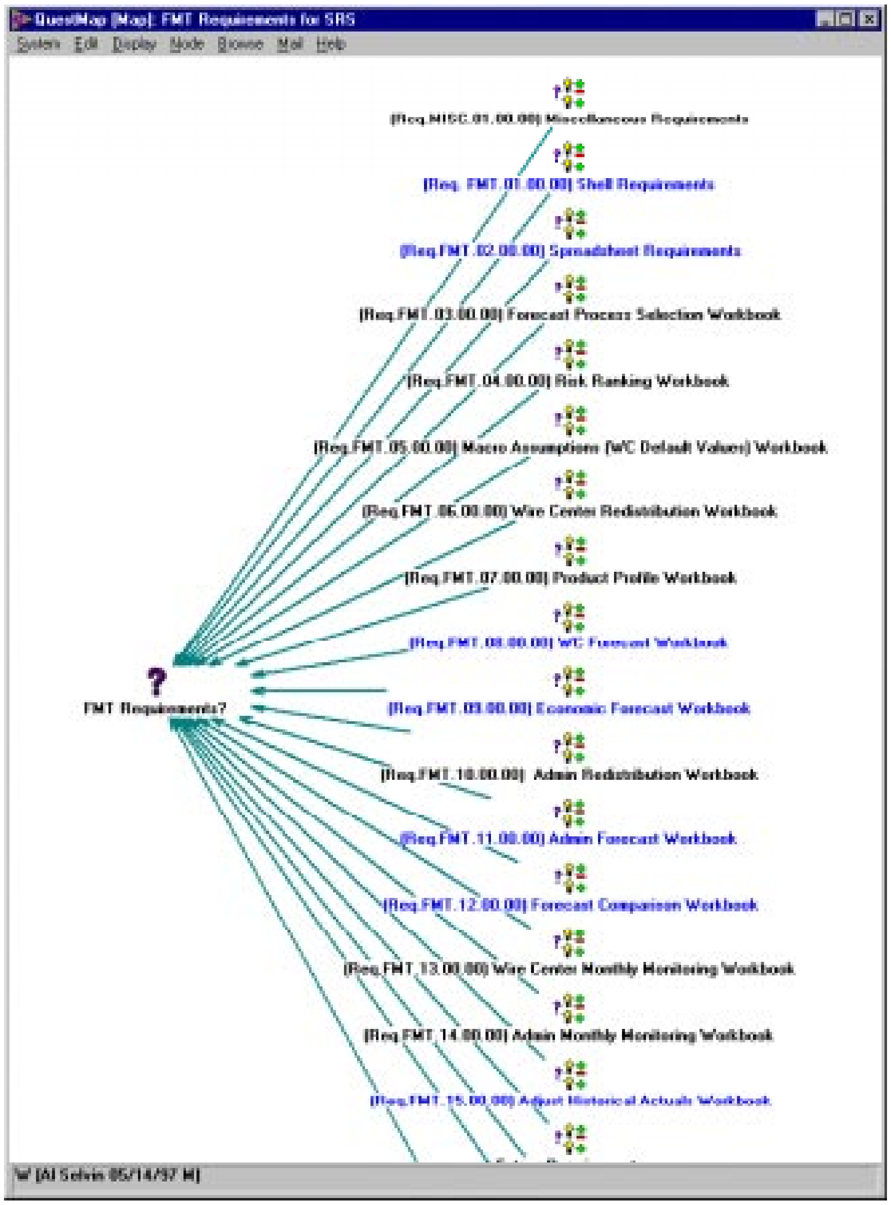
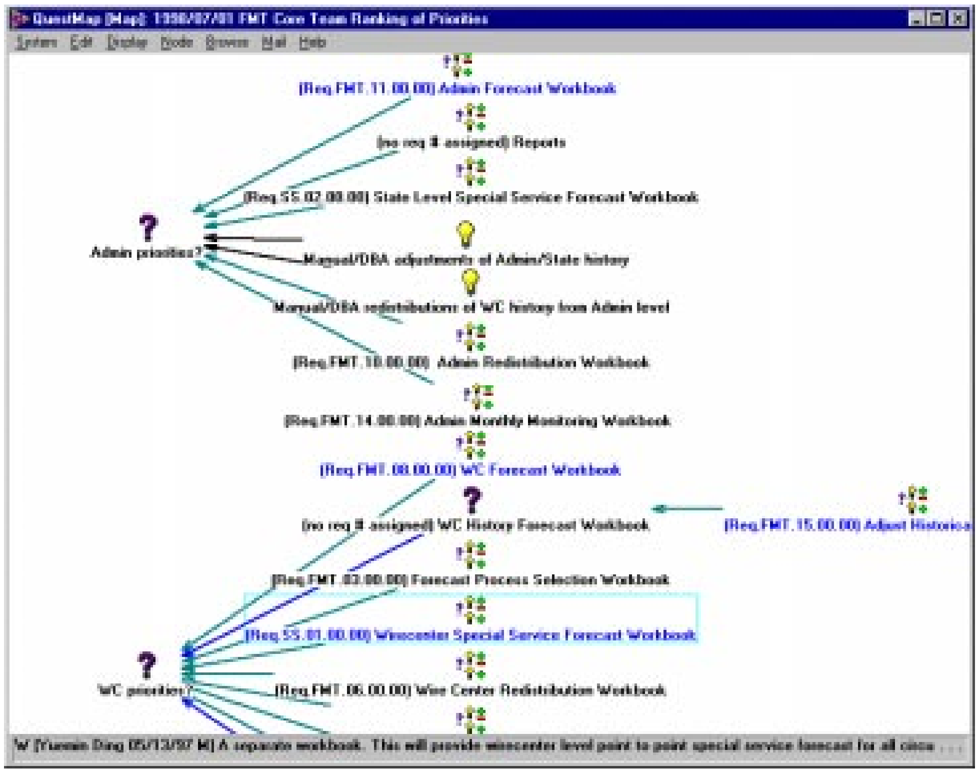

QuestMap [Map]: FMT Requirements for SRS
System Edit Display Node Browse Mail Help
[Req.MISC.01.00.00] Miscellaneous Requirements
[Req. FMT.01.00.00] Shell Requirements
[Req.FMT.02.00.00] Spreadsheet Requirements
[Req.FMT.03.00.00] Forecast Process Selection Workbook
[Req.FMT.04.00.00] Risk Ranking Workbook
[Req.FMT.05.00.00] Macro Assumptions (WC Default Values) Workbook
[Req.FMT.06.00.00] Wire Center Redistribution Workbook
[Req.FMT.07.00.00] Product Profile Workbook
[Req.FMT.08.00.00] WC Forecast Workbook
FMT Requirements?
[Req.FMT.09.00.00] Economic Forecast Workbook
[Req.FMT.10.00.00] Admin Redistribution Workbook
[Req.FMT.11.00.00] Admin Forecast Workbook
[Req.FMT.12.00.00] Forecast Comparison Workbook
[Req.FMT.13.00.00] Wire Center Monthly Monitoring Workbook
[Req.FMT.14.00.00] Admin Monthly Monitoring Workbook
[Req.FMT.15.00.00] Adjust Historical Actuals Workbook
W [Al Selvin 05/14/97 M]

Figure 2. Overview of requirements specifications (represented as map nodes)

Each of the map nodes could be opened to reveal nodes representing the detailed requirements for that module (each node could itself be opened to display further information about the particular requirement). A document generated from the above map contained more than one hundred pages of textual requirements description.

Eighteen months after the original requirements had been developed, the project leader gathered the map nodes representing the discrete groups of requirements on a new map. This was used in an interactive session with the project’s Core Team of users and customers to prioritise development for the various modules of the system.

The map below shows one result of the work done in that session. With the map displayed in front of the group with an LCD projector, the project leader first facilitated a general discussion of issues pertaining to the various requirements up for consideration for inclusion in an upcoming software release. Following this discussion, the group manipulated the horizontal order of the icons representing the various modules to indicate the priority order for development. When necessary, the team opened individual map icons to display both the original requirements and any additions and modifications that had been made between the original creation of the requirements node and the mid-1998 meeting (Figure 3).


QuestMap [Map]: 1998/07/01 FMT Core Team Ranking of Priorities
System Edit Display Node Browse Mail Help
[Req.FMT.11.00.00] Admin Forecast Workbook
[no req # assigned] Reports
[Req.SS.02.00.00] State Level Special Service Forecast Workbook
Admin priorities?
Manual/DBA adjustments of Admin/State history
Manual/DBA redistributions of WC history from Admin level
[Req.FMT.10.00.00] Admin Redistribution Workbook
[Req.FMT.14.00.00] Admin Monthly Monitoring Workbook
[Req.FMT.08.00.00] WC Forecast Workbook
[no req # assigned] WC History Forecast Workbook
[Req.FMT.15.00.00] Adjust Historica
[Req.FMT.03.00.00] Forecast Process Selection Workbook
[Req.SS.01.00.00] Wirecenter Special Service Forecast Workbook
WC priorities?
[Req.FMT.06.00.00] Wire Center Redistribution Workbook

W [Yuemin Ding 05/13/97 M] A separate workbook. This will provide wirecenter level point to point special service forecast for all circu . . .

Figure 3. Map showing ranking of priorities

This allowed the members of the Core Team, many of whom had not been involved with the original analysis, to get immediate access to background and reference information. The group then held discussions about the requirements as they related to their present concerns, as well as performed the prioritisation task.

Morphing of template-based concept maps into DFDs and review documents

The following example shows development in progress. Although hypertext representations such as those shown in the previous section were judged as

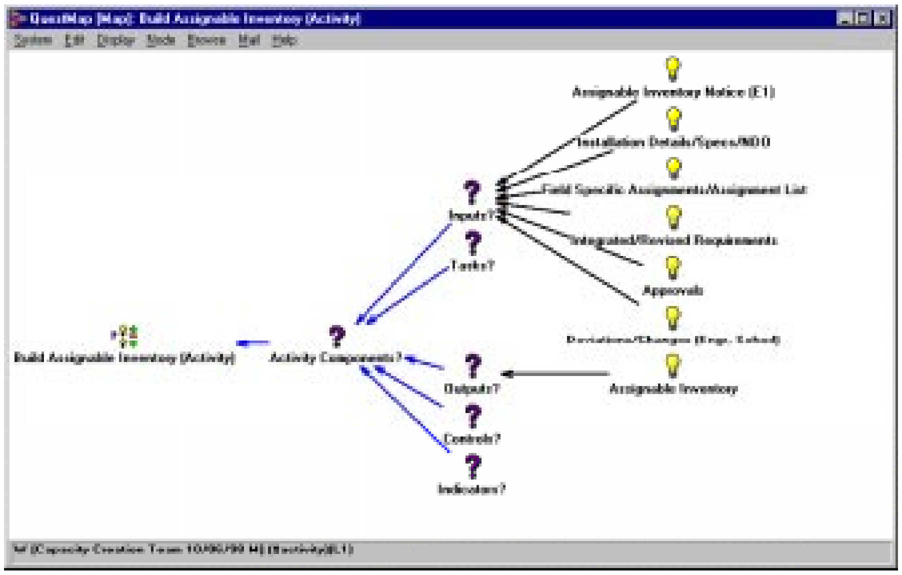

QuestMap [Map]: Build Assignable Inventory (Activity)
System Edit Display Node Browse Mail Help
Assignable Inventory Notice (E1)
Installation Details/Specs/NDO
Field Specific Assignments/Assignment List
Integrated/Revised Requirements
Approvals
Deviations/Changes (Engr. Sched)
Assignable Inventory
Inputs?
Tasks?
Build Assignable Inventory (Activity)
Activity Components?
Outputs?
Controls?
Indicators?
M (Capacity Creation Team 10/06/98 M) (#activity)(E1)

Figure 4. A process diagram from the Capacity Creation project

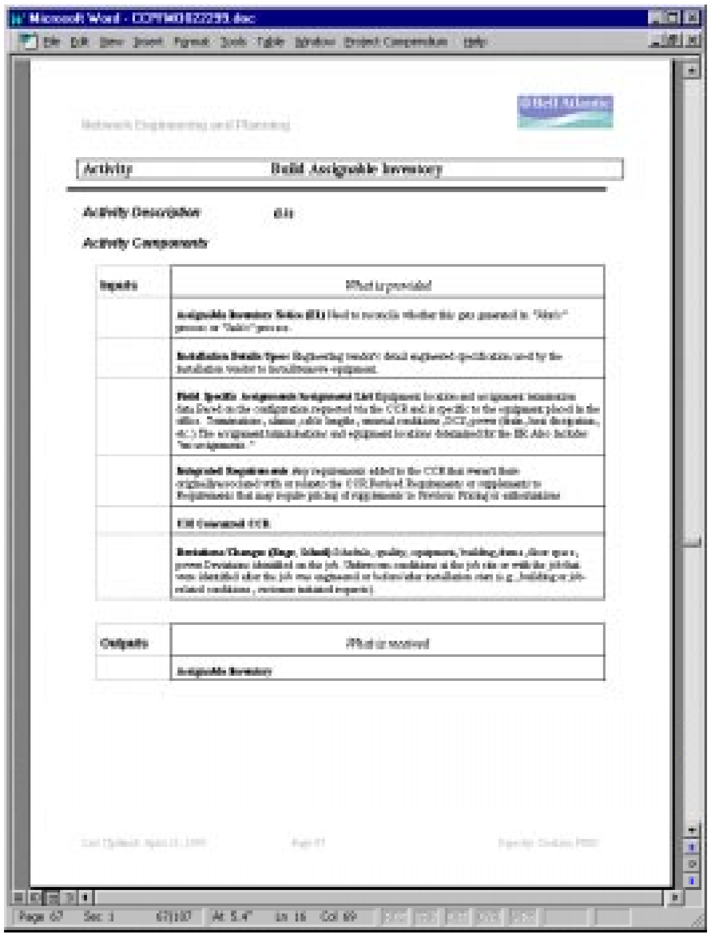
readable and effective by Project Compendium users[29], other user groups requested representations closer to those they were accustomed to. This was particularly true for software engineers and others experienced with structured representations such as DFDs. Project Compendium's developers are currently experimenting with software transformation of structured hypertext concept-map type representations into conventional DFDs.

In the following example, a process redesign team composed of representatives from various engineering departments created a map of one activity of a new design for the Central Office Capacity Creation process, a complex business process composed of more than seventy distinct activities (Figure 4). Elements depicted on the map fall into two categories: *questions* corresponding to a template developed by the facilitating project members, and *answers* gathered in collaborative sessions with participants drawn from many departments.

[29] For an evaluation, see Selvin, A., Sierhuis, M. *Experiences with Conversational Modeling: Building Models Through Collaboration*, NYNEX Science & Technology Technical Memorandum TM96-0045, White Plains, NY. Fall 1996.

The nodes representing answers were themselves drawn from “browser lists” of answers that other sub-teams gave to similar template-based models in other sessions. This concept-map type representation, while useful to the analysis teams working in group sessions, was not the best representation for two other communities involved in the effort. These included teams of managers approving the redesign, who were more comfortable reviewing “books” in a familiar format, as well as software development teams accustomed to working with DFDs.

Project Compendium already supports generation of customisable document formats generation of both formats without additional user work or time delay. Figure 5 shows a portion of the document generated from the map in Figure 4. Versions of this document were generated for each review cycle in the project and reviewed by the project’s several dozen participants.



Microsoft Word - CCPFM0822255.doc

Network Engineering and Planning

Activity	Build Assignable Inventory

Activity Description — [illegible]

Activity Components

Inputs	What is provided
	Assignable Inventory Notice (EI) [illegible]
	Installation Details Spec [illegible]
	Field Specific Assignments Assignment List [illegible]
	Integrated Requirements [illegible]
	CCR Generated CCR
	Deviations/Changes (Engr, Sched) [illegible]

Outputs	What is received
	Assignable Inventory

Page 67 Sec 1 67/107 At 5.4" Ln 16 Col 69

Figure 5: Automatically generated document in user-supplied format

Figure 6 shows a prototype example of automatic generation of a data flow diagram from the same concept map. Questions and other material unnecessary for this type of representation are abstracted away. The software recognises specially designated components of the template and creates diagram elements according to its predefined schema for data flow diagrams).

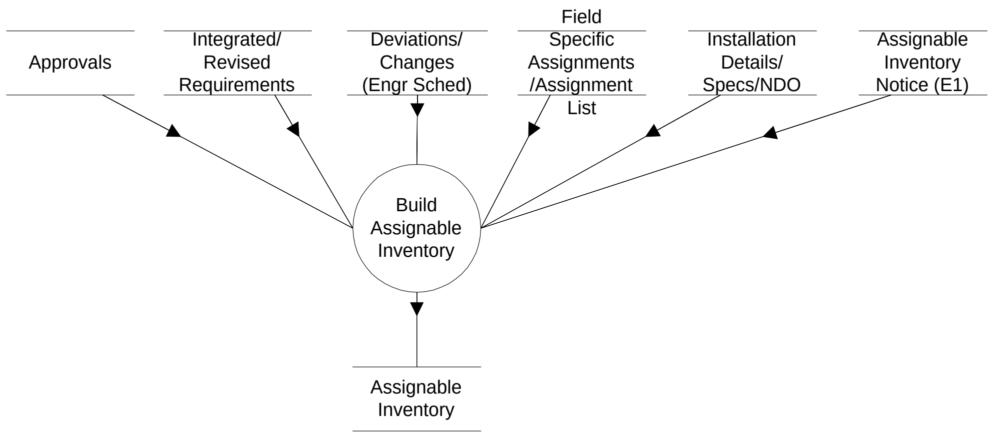

Approvals
Integrated/ Revised Requirements
Deviations/ Changes (Engr Sched)
Field Specific Assignments /Assignment List
Installation Details/ Specs/NDO
Assignable Inventory Notice (E1)
Build Assignable Inventory
Assignable Inventory

Figure 6. Automatically generated Data Flow Diagram

Discussion and Future Work

In each of the project examples described above, participants faced obstacles of community-specific literacies and decentered communication. Communities involved included software developers, managers and end users from business units affected by the systems under design or development, process redesign specialists, and attendees from other departments with an interest in the proceedings. In all cases, the requirements efforts took place in an environment of external change, organisational restructuring, and shifting responsibilities. Participants had to develop or work with requirements within constricted timeframes, with often conflicting directions from their respective departments, and with insufficient time to develop deep understanding of each other's frames of reference. In such environments, project facilitators had to provide requirements capture and display mechanisms that allowed participants to validate the representations on the fly, as well as to make results of individual sessions available to – and readable by – other participants. The need to provide a high degree of collaborative sense-making under difficult circumstances mandated that facilitators develop, augment, and publish representations serving community-appropriate representational literacies in rapid time.

The examples above show several ways in which Project Compendium's approach to software-assisted representational morphing can provide representations appropriate to particular participants' preferences, without requiring additional manual effort. In each of the examples, knowledge elements created in one time and use context were "repurposed" in both form (morphing from one representational form to another) and content (the ways in which individual knowledge elements were related to one another in the original context were changed to reflect the concerns and people in the second context. In each case, the original form and content were also preserved, and available for reference and/or augmentation at any moment.

Such transformations are critical in environments requiring rapid articulation and synthesis of requirements information across participating communities. There is a critical need to repurpose knowledge elements quickly and effectively across teams without requiring rework or "re-inventing the wheel." Currently, for example, Bell Atlantic is using the approaches described above as part of its Year 2000 contingency planning effort, involving five separate Integrated Process Teams charting contingency strategies for the company's core business processes. Each team's work must necessarily build on all the others, but the project's tight deadlines leave no time for review and training sessions. Using an effective requirements repurposing approach allows the teams to both develop and cross-validate their information efficiently and effectively.

While rigorous analysis of the role Project Compendium's representational morphing and other techniques has played in the success of these and other requirements efforts has yet to be performed, participant responses, informal surveys, and internal customer demand for the approach indicates that Project Compendium's users ascribe a high degree of value to it. We believe that this is because Project Compendium provides a relatively "neutral medium" for the articulation of ideas. By this we mean that there is a 'good faith' effort to represent all that is said by members of the different communities; even if its relevance to the immediate issue is not obvious, it should be captured and made part of the shared display, and group memory that is constructed as a by-product. Conklin (Conklin, 1998) characterises the spectrum between *transcribing* what is said and *interpreting* it, which usually involves distilling. Distilling/shortening is acceptable as long as representational integrity (as judged by the idea's owner) is preserved. A representational sense-making tool should have the ability to represent issues and ideas even when they do not immediately fit the predefined format. This legitimises the posing—and representation—of sense-making questions (Dervin, 1983) such as “why are we doing this?” or “are we at the right level?”

Future work will include a deep contextual analysis of how representational morphing has aided collaborative sense-making in situated project team use of Project Compendium. This will include a study of how particular manipulations, transformations, and reuse of particular knowledge elements representing requirements served the sense-making needs of particular participants in a software development project at various moments in the project's lifecycle.

References

Bannon, L. J., & Kuutti, K. (1996). Shifting Perspectives on Organizational Memory: From Storage to Active Remembering. *Proc. HICSS’96: 29th Hawaii International Conference on System Sciences*, (Hawaii (Jan., 1996)). IEEE.

Barton, D. (1994). *Literacy: An Introduction to the Ecology of Written Word*. Oxford: Blackwell.

Bellotti, V., Blandford, A., Duke, D., MacLean, A., May, J., & Nigay, L. (1997). Controlling Accessibility in Computer Mediated Communications: A Systematic Analysis of the Design Space. *Human-Computer Interaction*, 12, (1)

Bellotti, V., Buckingham Shum, S., MacLean, A., & Hammond, N. (1995). Multidisciplinary Modelling In HCI Design...In Theory and In Practice. *Proc. CHI’95: Human Factors in Computing Systems*, (Denver, Colorado (May 7-11, 1995)), 146-153. ACM Press: New York.

Blomberg, J. L., & Henderson, A. (1990). Reflections on Participatory Design: Lessons from the Trillium Experience. (Ed.), *Proceedings of ACM CHI'90 Conference on Human Factors in Computing Systems* (pp. 353-359)

Bowers, J. (1991). The Politics of Formalism. In M. Lea (Ed.), *Contexts of Computer-Mediated Communication* (pp. 232-261): Harvester Wheatsheaf.

Bowker, G. C. (1997). Lest We Remember: Organizational Forgetting and the Production of Knowledge. *Accounting, Management and Information Technologies*, 7, (3), 113-138.

Buckingham Shum, S. (1997). Negotiating the Construction and Reconstruction of Organisational Memories. *Journal of Universal Computer Science (Special Issue on Information Technology for Knowledge Management),* 3, (8), 899-928. <http://www.iicm.edu/jucs_3_8/> Reprinted in: *Information Technology for Knowledge Management.* (Eds.) Borghoff, U.M. and Pareschi, R.,Springer-Verlag: Berlin, Heidelberg, New York, 1998, pp. 55-78.

Chin, G. J., & Rosson, M. B. (1998). Progressive Design: Staged Evolution of Scenarios in the Design of a Collaborative Science Learning Environment. *Proc. CHI 98: Human Factors in Computing Systems*, (Los Angeles, CA), 611-618. ACM Press: NY.

Coad, P., & Yourdon, E. (1991). *Object-Oriented Analysis.* : Englewood Cliffs: Prentice-Hall.

Conklin, J. (1996). Designing Organizational Memory: Preserving Intellectual Assets in a Knowledge Economy. Group Decision Support Systems, Inc., 1000 Thomas Jefferson Street, NW, Suite 100, Washington, DC 20007, U.S.A.

Conklin, J. (1998). VIMS: Visual Information Mapping. *Training Course*Group Decision Support Systems, Inc.<http://www.gdss.com/icl/VIMS.html>

Conklin, J., & Burgess Yakemovic, K. C. (1991). A Process-Oriented Approach to Design Rationale. *Human-Computer Interaction*, 6, (3&4), 357-391.

de Hoog, R., et. al. Applying the Common KADS Organization Model. KADS-II Consortium (ESPRIT Project P5248)

Dervin, B. (1983). An Overview of Sense-Making Research: Concepts, Methods and Results. *Annual Meeting of the International Communication Association*, (Dallas, TX (May)).
<http://communication.sbs.ohio-state.edu/sense-making/art/artdervin83.html>

Isenmann, S., & Reuter, W. (1997). IBIS: A Convincing Concept ... But a Lousy Instrument? *Proc. of DIS'97: Conference on Designing Interactive Systems: Processes, Practices, Methods and Techniques*, 163-172. ACM: New York.

Jacobson, I., et. al. (1992). *Object-Oriented Software Engineering: A Use Case Driven Approach*. Wokingham: ACM Press.

Kruizinga, E., & van der Spek, R. (1993). *Model-Based Development of Knowledge-Intensive Systems Workshop*, (Centrum voor Kennistechnologie, The Netherlands).

Kuutti, K. (1998). Supporting Perspective Making and Perspective Taking: A Framework for Storing Contextual Information for Reinterpretation. *Proc. 7th International Workshop on Hypertext Functionality (Organizational Memory Systems & HTF)*, (Helsinki, Dec. 12-13). University of Oulu, Dept. Computer Science Technical Report Series.

Leigh Star, S., & Greisemer, J. (1989). Institutional Ecology, "Translations," and Coherence: Amateurs and Professional in Berkeley's Museum of Vertebrate Zoology, 1907-1939. *Social Studies of Science*, 19, , 387-420.

Miller, D. S., John, Smith, G., & Muller, M. J. (1992). TelePICTIVE: Computer-Supported Collaborative GUI Design for Designers with Diverse Expertise. *Proceedings of the ACM Symposium on User Interface Software and Technology 1992*, 151-160. ACM Press: NY.

Muller, M. J. (1991). PICTIVE - An Exploration in Participatory Design. *Proceedings of ACM CHI'91 Conference on Human Factors in Computing Systems*, (New Orleans, USA), 225-231. ACM Press: NY.

Robinson, M., & Bannon, L. (1991). Questioning Representations. In L. Bannon, M. Robinson and K. Schmidt (Ed.), *Proc. of ECSCW'91: 2nd European Conference on Computer-Supported Collaborative Work* (pp. 219-233). Amsterdam Sept 25-27: Kluwer.

Selvin, A. (1998). Supporting Granular Reuse of Knowledge Elements in an Organizational Memory System. *Proc. 7th International Workshop on Hypertext Functionality (Organizational Memory Systems & HTF)*, (Helsinki, Dec. 12-13). University of Oulu, Dept. Computer Science Technical Report Series.

Selvin, A. (1999). Supporting Collaborative Analysis and Design with Hypertext Functionality. *Journal of Digital Information*, 1, (4) <http://jodi.ecs.soton.ac.uk/Articles/ v01/i04/Selvin/>

Suchman, L. (1993). Do Categories have Politics? The Language/Action Perspective Reconsidered. *3rd European Conference on Computer-Supported Cooperative Work*, (Milan, Italy (13-17 Sept.)), 1-14. Kluwer Academic Publishers.

Weick, K. (1993). Sensemaking and Group Support Systems. In L. Jessup and J. Valacich (Ed.), *Group Support Systems: New Perspectives*. New York: Macmillan.

Weick, K. E. (1995). *Sensemaking in Organizations*. Thousand Oaks, CA: Sage Publications.

THE DYNAMIC MODELS FOR SOFTWARE DEVELOPMENT PROJECTS AND THE MACHINE LEARNING TECHNIQUES

Isabel Ramos Román
José C. Riquelme Santos
Dpto. de Lenguajes y Sistemas Informáticos
Universidad de Sevilla (Spain)
email {isabel.ramos|riquelme}@lsi.us.es

ABSTRACT

During recent years the management of software development projects (SDPs) has reached significant advances. One of these progresses is the realization of dynamic models that permit us to model the complex behaviour of the software development processes. The main advantage of these models is the possibility to experiment before the execution of the project. In this way we can see the effect that the application, or non application, of different management policies will have on the project. In this paper we present a part of the results obtained by combining, on one hand, the use of a tool that learns producing rules, and additionally a dynamic model of SDP. This permits us to obtain management rules applicable to a SDP for estimating good results with the variables that the project manager desires.

INTRODUCTION

Traditionally, the known problem of software crisis has been treated from the view point of the technology of development used. Therefore, significant advances have been achieved in the use of new methodologies of development, CASE tools, reusing source code, etc. With the appearance, during recent years, of the dynamic models for SDPs, progresses have been reached in the field of project management tools and in the advising of the forever complex process of decision making.

The simulation of a dynamic model for a SDP permits us, before beginning the development, to find out what impact a change of technology would have on the project [Chichakly 93], the application, or non application, of different management policies and the maturity level of the very development organization. In this paper we shall present some of the results from the tasks that we are performing in order to obtain management rules[30] for the SDPs . The knowledge of these management rules can be obtained before the beginning the project's execution and it will permit us to obtain good results for the variables (delivery time, cost, quality, productivity, etc.) that the project manager desires. Furthermore, the management rules will permit us to analyse which of the management policies are more significant for securing the defined initial objectives in the SDPs, as well as to recognize if these policies can be applicable or not.

In order to obtain the management rules, we have combined the advantages that a system that learns based on rules presents and the information that a dynamic model for SDPs provides. In the following sections, we first present a brief introduction into the concept of machine learning and the tool used for this purpose; later, we present the information that is given to the dynamic system for SDPs. Finally, we apply these methods to a specific SDP.

MACHINE LEARNING

The computational techniques and tools designed to support the extraction of useful knowledge from databases are traditionally named machine learning. More recently the names of data mining or Knowledge Discovery in Databases are used (KDD). In general, the previous techniques try to extract, in an automatic way, information useful for decision support or exploration and understanding the phenomena that is the data source.

A standard KDD process is constituted by several steps [Fayyad 96] such as data preparation, data selection, data cleaning, data mining and proper interpretation of the results. Therefore, data mining can be considered a particular step that consists in the application of specific algorithms for extracting patterns from data. A wide variety and number of data mining algorithms are described in the literature from the fields of statistics, pattern recognition, machine learning and databases. Most data mining algorithms can be viewed as compositions of three

[30] We call management rule to a set of management policies (decisions) that to take the manager for carrying out the project final objectives.

basic techniques and principles: the model (classification, regression, clustering, linear function, etc.), the preference criterion, usually some form of goodness-of-fit function of the model to the data and search algorithm (genetic, greedy, gradient descent, etc.).

Thereby, the choice of a method of data mining depends on the model representation that we need. Given that our objective is to find rules to describe the behaviour of a SDP, our election has been to work with decision trees. A decision tree is a classifier with the structure of a tree, where each node is a leaf indicating a class, or an internal decision node that specifies some test to be carried out on a single attribute value, and one branch and subtree for each possible outcome of the test. The main advantages of decision trees are their utility for finding structure in high-dimensional spaces and the conversion to rules easily meaningful for humans is immediate. However, classification trees with univariate threshold decision boundaries which may not be suitable for problems where the true decision boundaries are non-linear multivariate functions.

The decision tree algorithm more spread is C4.5 [Quinlan 93]. Basically, C4.5 consists in a recursive algorithm with divide and conquer technique that optimises the tree construction on basis to gain information criterion. The program output is a graphic representation of the found tree, a confusion matrix from classification results and an estimated error rate. C4.5 is very easy to set up and run, it only needs a declaration for the types and range of attributes in a separate file of data and it is executed with UNIX commands with very few parameters. The main disadvantage is that the regions obtained in continuous spaces are hyperrectangles due to the test of the internal nodes are the forms: $p_i \geq L$ or $p_i \leq U$. However, given our purpose in this work the results supplied by the C4.5 are perfectly valid.

DYNAMIC MODEL FOR SOFTWARE DEVELOPMENT PROJECT

To obtain the database that is the entry of the C4.5, we have used a dynamic model for SDP proposed in [Ramos 98a] and implemented in the environment simulation Vensim®. The variables that permit to know the basic behaviour of a dynamic system are defined through differential equations. Furthermore, the model possesses a set of parameters that define different behaviours. The values of the parameters can be chosen randomly in an interval defined by the user, being the model simulated below. Such a record for the database is generated with the values of the parameters and the values obtained for the system variables that are desired. Starting from this generated database, the C4.5 learns examining the supplied data and proposing a set of rules for the decision making.

As previously commented, the dynamic model for SDPs includes a set of parameters that permit us to study different behaviours. These are provided by the management policies that can be applied in the SDPs, both related to the environment of the project (initial estimations, complexity of the software, etc) and the related to the development organization (personnel management, effort assignment, etc.) of and its maturity level (like the average delays through the realization of the activities of detection and correction of errors).

OBTAINING OF MANAGEMENT RULES FOR SDPs.

The utilization of a tool for the automatic obtaining of management rules permits the management of SDPs to face different situations, when they have to define the management policies more adequate to optimize the final values of determined variables, separated (delivery time, cost, quality, productivity, etc) or united (delivery time, cost and quality simultaneously).

Once the management rules have been obtained, it is the manager of the project who decides which rule or rules are the easiest to apply, in function of the specific project and of the software organization of the one which is working. He/she will also be the one which, before beginning the development, and once the parameters that appear in the obtained management rules have been analyzed, has to decide which parameters can be operated to maintain it within the values that appear in the rules and which not, to optimize the results of the variables or groups of variables in his interest. In any case, he/she will also realize that if he/she doesn't keep within the obtained values, the said optimization will not be guaranteed.

In view of the results obtained in the following sections and of the complexity that the management and control of a SDP conveys, we propose at least two basic criteria in the election of management rules that are going to be applied: first, to choose rules that use parameters that are easy to control and modify and, in second place, if it is possible, to choose rules with a small number of parameters. In the following sections we will use the data of a real SDP proposed in [Abdel-Hamid 91] which we will call "PROJECT". In section 4.2, we will obtain management rules that make "GOOD" the final values of the delivery time and of the cost of the project (measured in effort units), separated (if the priority objective of the project director is to optimize some specific variable at "any price") or simultaneously. To show the usefulness that the obtaining of management rules presents, in section 4.3, management rules have been obtained that permit us to accomplish a post-mortem analysis of the PROJECT, that is to say, how we would have been able to improve the final results of this project.

Entry data

From among the parameters that permit us to define the development environment, so much for the project as for the organization, and the maturity degree of the development organization, we have collected, by considering them representative of each one of the blocks cited to the final of epigraph 3, those which appear in Table 1. Indicated in this table are, for each parameter, the name that it has in the Basic Dynamical Model [Ramos 98a], the interval values that it can take, a brief description of the meaning that it has and the units of measurement. It is considered, for the specific SDP that we are going to analyze, that the rest of the parameters [Abdel-Hamid 91] are not going to vary.

NAME	INTERVAL	DESCRIPTION (UNITS)
DEDIC	(20 - 100)	Average dedication of the technical personnel (dmnl).
RESQA	(5 - 15)	Average delay in the development of Quality activities (days).
READE	(20 - 120)	Average delay in the appropriateness of the new technical personnel in the project (days)
RECON	(1 - 40)	Average delay in accomplishing the contracting of technical personnel (days).
PORTE	(30 - 100)	Percentage of technicians at the beginning of the project in relation to the estimated average value (dmnl).
TECCO	(1 - 4)	Technicians to contract for each experienced full time technician (technicians) .

POMAX	(0 - 300)	Maximum percentage permitted in delivery time (dmnl).
RENOT	(5 - 15)	Average delay in notifying the real state of the project (days).
ESFPR	(0,1 - 0,25)	Nominal effort necessary in the Tests stage by error (technicians-day).
RETRA	(1 - 15)	Average delay in the transferring of technical personnel that exceed to other projects (days).
POFOR	(10 - 40)	Average percentage of the experienced technicians dedication to training (dmnl).
INTAM	(0 - 50)	Initial underestimation of the project's size in source code lines (ldc).

Table 1: Representative parameters of the project's environment and of the organization's environment.

The variables that have been studied in this section are the cost and the delivery time of the project. The intervals defined for the variables delivery time and cost of the project are the following:

Delivery time (days): The values in the interval (320 - 384), corresponds to the delivery time values understood to be between the initial estimated time and a postponement margin of 20% on the initial estimate. These values have been labelled as "GOOD". Delivery time values superior to that of 20 % on the initial estimate have been labelled as "BAD".

Cost or effort (technicians - day): The values in the interval (1.111-1.444), corresponds to final effort values understood to be between the initial estimated effort and an amplification margin of the same of 30% on the initial estimate. These values have been labelled as "GOOD". The values of the final effort superior to that of 30 % on the initial estimate have been labelled as "BAD".

Management rules obtained before the beginning of the PROJECT's execution (prior analysis)

In this section, we consider that the PROJECT has not yet begun its execution and, by so much, we want to know if management rules exist that permit us to estimate "GOOD" results for the delivery time and the cost of the project.

Below, we will analyze, in first place, the management rules that make GOOD the delivery time and the PROJECT's cost, this case have been named CASE 1: POMAX can oscillate between 0 % and 20% in relation to the initial estimated time, which is considered as a fixed term policy to moderate term and furthermore we find ourselves in a development environment where the personnel management policy is rapid, that is to say: READE varies between 20 and 30 days, RECON varies between 1 and 10 days and RETRA varies between 1 and 5 days (this criterion may vary depending on the organization).

Management rules obtained in the CASE 1 to obtain "GOOD" results for the cost and the delivery time of PROJECT delivery, before beginning the execution of the same, are shown in table 2[31]:

In total, 8 management rules have been obtained. A great number of cases have been found (250 of 300 accomplished simulations) in those which values considered as "GOOD" for the delivery time are obtained. The foregoing means that a rapid personnel management policy favours the finalization of the project within some reasonable delay periods [Ramos 98b]. A general reading of the obtained rules (Table 2) indicates to us:

[31] Of the decision tree given by C4.5, in this work, we show only the management rules that permit us to estimate GOOD results.

DEDIC <=0,27; TECCO >3,88	**(1)**
2,3 < TECCO < =3,88; PORTE > 0,44	**(2)**
0,27 < DEDIC <= 0,33; TECCO < =1,46; PORTE > 0,36	**(3)**
DEDIC > 0,47; TECCO < =1,46; PORTE > 0,47	**(4)**
DEDIC > 0,74; TECCO < =1,46; 0,36 < PORTE <= 0,47	**(5)**
0,27 < DEDIC <= 0,33; TECCO >1,9; POFOR > 0,14	**(6)**
DEDIC > 0,44; TECCO >1,46; PORTE > 0,34	**(7)**
0,33 < DEDIC < = 0,44; TECCO >2,24; PORTE > 0,34	**(8)**

Table 2: Management rules that permit to estimate "GOOD" results for the delivery time of PROJECT.

That the most important parameters to control the delivery time within the demanded limits, are directly related to the effort assignment (average dedication of the technical personnel in the project) and the personnel management (new technical personnel for each experienced full time technician and estimated percentage of technical personnel at the beginning of the project).

The rules provide us with the values range in which the said parameters should be moved: for different values of the personnel's average dedication in the project, we will be able to obtain the objectives on the delivery time controlling, within the intervals of values indicated in the rules, the number of new technical personnel for each experienced full time technician and estimated percentage of technical personnel at the beginning of the project.

The reading of management rules (1) and (2) of Table 2 indicates to us that if:

The dedication of the technical personnel in the project (DEDIC) is lesser or equal to 27 %, we will be able to obtain values considered as "GOOD" for the delivery time of the project if the number of new technical personnel for each experienced full time technician (TECCO) is greater than 3,88 or if it is understood to be between values greater than 2,3 and lesser or equal to 3,88 and the percentage of estimated technical personnel at the beginning of the project (PORTE) is greater than that of 43 %".

On the other hand, rule (5) of Table 2, proposes that if:

"The average dedication of the technical personnel (DEDIC) is greater than 74 % the objectives of delivery time will be achieved if the number of new technical personnel for each experienced full time technician (TECCO) takes lesser than, or equal values of 1,46 and the estimated percentage of technical personnel at the beginning of the project (PORTE) takes greater values of 36 % and lesser than, or equal to that of 47 %".

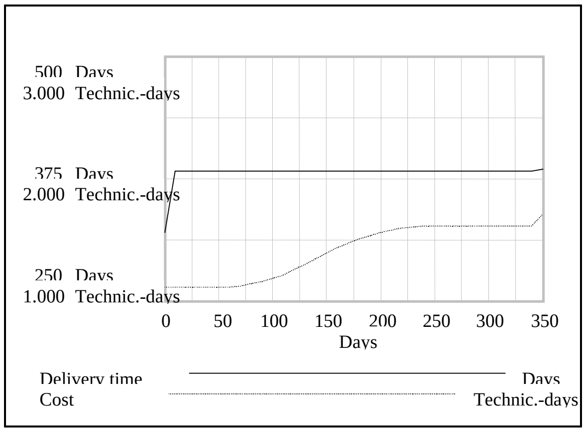


Fig. 1: Evolution of the delivery time and the PROJECT's cost upon applying the rule (1).

In Figures 1 and 2, we can check the results that we would obtain for the delivery time and the PROJECT's cost, if we apply management rules (1) or (5).

As was expected if we apply management rule (1), in which the dedication of the technical personnel is very low (lesser than, or equal to that of 27 %), and management rule (5), in which the dedication is high (greater than 74%), we will obtain that the delivery time of the project with respect to the initial forecast which would be 20 % and increased by almost 11 % respectively. While the final values of the cost would surpass the initial forecast by 53 % and 58 % respectively. That is to say, as was expected, if we apply these management rules we would obtained values considered as "GOOD" for the delivery time of the project but not for the cost. The decision of applying one or the other management rule will be taken by the project manager.

It is interesting to observe in Figure 1, the jump that is produced in the initial estimate of the delivery time, practically at beginning of the project. With the initial estimates imposed to the project, especially what is referred to the dedication of the technical personnel (which takes values considered as very low), was foreseeable the difficulty in maintaining the delivery time within the demanded limits, therefore seems timely to modify the initial forecasts of the delivery time as soon as possible to avoid future modifications [Ramos 98c].

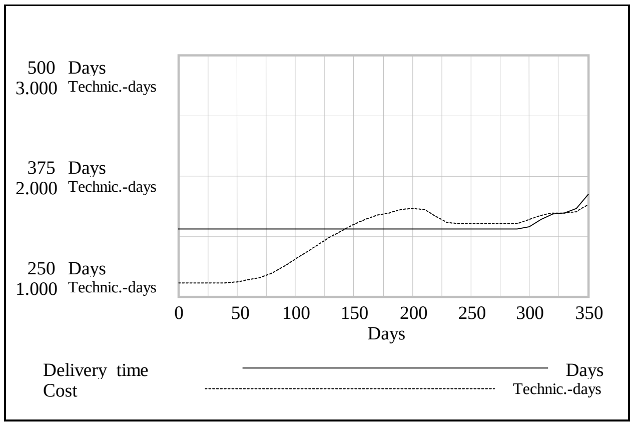


Fig 2: Evolution of the delivery time and the PROJECT's cost upon applying the rule (5).

For CASE 1, we have only obtained one management rule that permits us to estimate values considered as "GOOD" for the PROJECT's cost. This will be possible if (Table 3):

"The dedication of the technical personnel (DEDIC) takes values lesser or equal to 43 %, the estimated percentage of technical personnel at the beginning of the project (PORTE) is lesser or equal to 34 % and the activities of Qualit (RESQA) are accomplished in a time greater than 7 days".

PORTE<=34; DEDIC < = 0,43; RESQA > 7 (1)

Table 3: Management rule that permits us to estimate "GOOD" results for the PROJECT's cost.

As previously mentioned, the application of the rule obtained in Table 3 does not guarantee the fulfilment of the initial objective of the project's delivery time. We can prove how upon applying the obtained management rule, the effort or cost to accomplish the PROJECT will increase by 21%, approximately, with respect to the initial estimate. But not the delivery time, which will surpass the initial estimate by 43 %.

As for the previous results, our interest would be centred in finding the management rules that permit us to obtain, simultaneously, values considered as "GOOD", for the delivery time as well as for the PROJECT's cost in CASE 1.

For the aforementioned, we must say that cases have not been found in which both variables give values simultaneously labelled as "GOOD". The previous result, still being distressing for a project's director, provides valuable information: with the initial conditions imposed to the PROJECT, it will not be possible to simultaneously obtain "GOOD" results for the time and the cost of the project.

Management rules obtained once the PROJECT has ended (post-mortem analysis)

On the other hand, the real values of the delivery time and the PROJECT's cost were 387 days and 2.092 technicians-days respectively [Abdel-Hamid 91]. These values were considered by us, in the previous section, as "BAD" since they surpassed the initial time and cost estimates by 20% and 30 % respectively. Therefore, before these final results, and if it had been possible to obtain management rules, we would have to ask ourselves: Do management rules exist that might have improved the obtained final results?.

In response to the previous question, we are going to vary the intervals defined for each one of the analyzed variables:

Delivery time (days): Values in the interval (320-387) have been labelled as "GOOD" by being inferior to the obtained real results. Values greater than 387 was considered as “BAD”.

Effort or cost (technician - days): The values in the interval (1.111 - 2.092) have been labelled as "GOOD" by being inferior to the obtained real results. Values greater than 2.092 was considered as “BAD” by surpassing the obtained real results.

Below, with the previously defined intervals for PROJECT, we will analyze CASE 2: the delivery time policy is maintained within the initially defined limits, but the policy of personnel management encompasses the three possible cases (slow, average, rapid), that is to say: READE varies between 20 and 120 days, RECON varies between 1 and 40 days and RETRA varies between 1 and 15 days.

With this assumption, we want to know if by combining different alternatives of personnel management policies, we could have applied management rules that might have improved, simultaneously, the delivery time and the PROJECT's cost. In fact, we want to know: What values the parameters must have taken to improve the obtained real results?, and a second question that the actual development organization must answer is, Would these obtained values have been easy to modify?.

The management rules obtained for CASE 2 are shown in Table 4. As can to see, 5 management rules have been obtained. Management rules (1) and (2), for example, indicate to us that the final results achieved for the delivery time and the PROJECT's cost could have been improved either if (rule 1):

READE <=27;

RENOT < = 12; INTAM > 0,40, DEDIC > 0,6
(1)

RENOT > 12; RETRA > 10
(2)

READE > 27;

RETRA < = 14;

INTAM < = 0,47; POFOR <= 0,13;
ESFPR > 0,22 (3)

INTAM > 0,47; POFOR <= 0,18
(4)

RETRA > 14; DEDIC > 0,82
(5)

Table 4: Management rules that permit us to estimate "GOOD" results, simultaneously, for the delivery time and the PROJECT's cost.

"The integration of the new personnel in the project (READE) might have been lesser than, or equal to 27 days and the notification of the progress of the project (RENOT) might have been lesser than, or equal to 12 days and the initial underestimation of the size of the product in source code lines (INTAM) might have been greater than 40 % and the dedication of the technical personnel in the project (DEDIC) might have been greater than 60 %".

or if (rule 2):

"The integration of the new personnel in the project (READE) might have been lesser than, or equal to 27 days and the notification of the project's progress (RENOT) might have been (greater than 12 days and the transfer of the technical personnel to other projects (RETRA) might have been greater than 10 days".

Therefore, based on the previous management rules, we can answer the first of the questions that we previously mentioned. The answer would be: yes, PROJECT's final results could have been improved and the values of the parameters appear in the management rules of Table 4. The second question can only be answered by the actual development organization.

CONCLUSIONS AND FUTURE WORKS

The obtaining of management rules for SDPs can be applied before beginning the execution of a project to define the management policies more adequate for the project which is going to be accomplished. It can also be used in projects already ended to accomplish a post-mortem analysis. The usefulness of these rules consists in the possibility of:

- To obtain values considered as good (acceptable or bad) for anyone of the variables that result interesting to analyze, either in an independent way or simultaneously with other variables.

- To analyze which are the parameters implicated in the definition of management policies and level of maturity of the organization and which are easy to modify.

- To study which of the previously mentioned parameters are those which more influence the good results obtained.

- To analyze for a same SDP a great number of possible situations.

In fact, we can say that it is possible to obtain automatically management rules for a SDP and to recognize what are the management policies that guarantee the attainment of its objectives.

In light of the potential that the obtaining of management rules presents from a dynamic model, our future projects are guided in the application of fuzzy logic techniques and in the creation of a tool that permits the manager of a SDP to automatically generate rules for the variables and values that he desires.

BIBLIOGRAFÍA

[Abdel-Hamid 91] Abdel-Hamid, T.; Madnick, S.: "Software Project Dynamics: an integrated approach", Prentice-Hall, 1991.

[Chichacky 93] Chichacly, K. J.: "The bifocal vantage point: managing software projects from a Systems Thinking Perspective". American Programmer, pp.: 18 - 25. May 1993.

[Fayyad 96] Fayyad, U.; Piatetsky-Shapiro, G.; Smyth P.: “The KDD Process for Extracting Useful Knowledge from Volumes of Data”. Communications of the ACM. Vol. 39, Nº 11, pp.: 27-34. November 1996.

[Quinlan 93] Quinlan, J.: “C4.5: Programs for Machine Learning”, Morgan Kaufmann Pub. Inc., 1993.

[Ramos 98a] Ramos, I.; Ruiz, M.: “A Reduced Dynamic Model to Make Estimations in the Initial Stages of a Software Development Project”. INSPIRE III. Process Improvement through Training and Education. Edited by C. Hawkings, M. Ross, G. Staples, J. B. Thompson. Pp.: 172 - 185, September 1998.

[Ramos 98b] Ramos, I.; Ruiz, M.: “Aplicación de diferentes políticas de contratación de personal en un proyecto de Desarrollo de Software”, IV Congreso Internacional de Ingeniería de Proyectos, pp. 195-202, Cordoba, Noviembre 1998.

[Ramos 98c] Ramos, I.; Ruiz, M.: “Análisis del impacto de la política de dedicación del personal técnico sobre un Proyecto de Desarrollo de Software”, IV Jornadas de Informática, pp. 429-438, Las Palmas de Gran Canaria, Julio 1998.

Improving the Requirements Definition: The RESPECT Project

F. Calzolari and E. Cozzio

ITC-Irst, I-38050 Povo (Trento), Italy
tel. +39 0461 314583, fax +39 0461 314591
e-mail: calzolar@irst.itc.it

Federazione Trentina delle Cooperative
Via Segantini, 10 - I-38100 Trento, Italy
tel. +39 0461 898320, fax +39 0461 895431
e-mail: enrico.cozzio@ftcoop.it

Abstract

The software industry has to cope with the rapid technological evolution and the global market competition, in order to satisfy the growing user demands for quality, services and bandwidth.
Experience in developing systems has shown that an inadequate understanding of system requirements is the single most important cause of user dissatisfaction and system failure. Therefore, once expectations are better understood, product quality is expected to improve accordingly.
As it happens for the most part of small companies, the FTC (FTC stands for the Trentino Federation of Cooperatives) software development process is actually largely informal and deadline driven. As a consequence, the process is deficient in collecting user expectations, addressing it in only an ad hoc way. The RESPECT project addresses this problem, providing a way to move from an informal and unsupported software development process to a more formal one, adopting new methodologies and applying suitable tools.
The objective of the RESPECT Project is to improve the requirements' specification and analysis phase by formalizing the process of requirement capturing and by adopting a CASE tool to support this phase.
This paper summarizes the project experience, highlighting some of the steps to make a small organization evolve from an informal development process to a better defined and more formal one.

1 Introduction

In the software industry changes force companies to cope with the rapid technological evolution and the global market competition, in order to satisfy the growing user demands for quality, services and bandwidth. Moreover, software systems become increasingly complex and entail ever growing costs from reworked requirements [8].

In fact, experience in developing systems has shown that an inadequate understanding of system requirements is the single most important cause of user dissatisfaction and system failure [12]. Therefore, once expectations are well understood, product quality is expected to improve accordingly [5].

As it happens for the most part of small companies, the FTC (FTC is the acronym from the italian words for the Trentino Federation of Cooperatives) software development process is actually largely informal and deadline driven. As a consequence, the process is not supported by a rigorous model, showing as a point of weakness the collecting of user expectations, addressing it in only an ad hoc way.

When FTC foresaw a market expansion from the internal company market to the external one, the software development department felt they had to improve the overall software development process, starting from the first step: the requirement specification and analysis phases.

The above business objective implies the technical objectives of the Process Improvement Experiment RESPECT, which is aimed at automating the requirement specification phase, increasing requirement stability and reducing the number of user-analyst cycles.

The RESPECT project is an ESSI Process Improvement Experiment funded by the European Commission under the Esprit contract 27713. RESPECT involves both the software development group in FTC (the prime contractor) and the ITC-Irst Software Engineering Group that acts as consultant for all methodological aspects of introducing new methodologies and tools.

Moving from the current practice, that is largely unformalized and almost completely unsupported, RESPECT is introducing new methodologies and automatic CASE tools to support the requirement specification and analysis phase, in order to decrease the overall development effort and increase the achieved software quality.

The experiment is being performed by defining a new requirement specification process that includes the use of automatic tools (Rational RequisitePro and IBM Visual Age) to be applied to the selected baseline project and by measuring the benefits obtained in terms of higher customer satisfaction, reduced effort per requirement, reduced time to market, reduced rework and higher software quality. An existing development team will employ such new techniques to the baseline project, comparing them against their own past experiences with a traditional methodology (i.e., comparing the situation to the existing development process).
This paper summarizes the project experience, highlighting how a small organization can evolve from an informal development process to a better defined and more formal one. It is organized in four Sections. Section 2 describes the partners involved in this project and then the project itself, outlining the approach and the expected benefits. Section 3 is devoted to the main project's activities performed so far. Tools screening and selection based on tools' features (Section 3.1), the definition of requirements guidelines (Section 3.2) and training activities (Section 3.3) will be presented. In addiction, Section 3.1.1 will present the DESMET project: whenever some new tool or methodology is to be evaluated, several factors add up to the purely technical ones: the influence of human factors and sociological issues, that can bias the evaluation exercise. In order to guide tools screening and selection we adopted the DESMET methodology: Section 3.1.1 schematically shows some results obtained by applying the DESMET approach.
Finally, in Section 4 conclusions will be drawn.

2 The two project partners

2.1 The Federazione Trentina delle Cooperative

The Federazione Trentina delle Cooperative s.c.r.l. (FTC) is a non-profit association, organized as a consortium among cooperatives operating in the area of Trento, a small town in the heart of the Italian Alps. This consortium legally represents its associated members as well.
By now FTC members are: 83 credit cooperative banks, 136 agricultural cooperatives (e.g. wine producers cooperatives, fruit and vegetable producers consortia, zootechnic cooperatives), 295 cooperatives working both in innovative and emerging fields (e.g. environmental and protection

services, social cooperatives) and in traditional industrial production and services (e.g. Insurance Companies, Software Houses).
The statutory goals of FTC spread from legal assistance to accounting and management support. FTC tasks range from auditing to management and marketing services, to promoting cooperation, to public relations and advertising, to educational and training programs, and to software production.
The complex and multi-purpose activity of FTC requires appropriate informatic support, provided by the software development group of the FTC, whose tasks span from developing software, to maintaining existing programs, also providing on site interventions and support for networking.
The aim of the experiment is to demonstrate the improvement of FTC software development process by showing that the adoption of automatic tools in the requirement specification and analysis phase enables FTC software involved people to decrease overall development effort and increase software quality.

2.2 The ITC-Irst

ITC-Irst, the RESPECT subcontratctor, is a public research institute whose activities include software engineering and maintenance. ITC-Irst holds a solid background in software engineering issues, especially in Object-Oriented modeling [2][3], effort estimation [7][6], static and dynamic code analysis [13][14] as well as in software metrics [4]. Several tens of articles presented at international conferences or published by scientific journals the impact of such activities on the scientific community.
Within the RESPECT project, ITC-Irst cooperates with the FTC for scientific and methodological aspects, supporting activities regarding tool selection, customization and training, implementation as well as requirement process and guidelines definition.

2.3 The Project's Work Packages

The experiment is performed by defining a new requirement specification process that includes the use of automatic tools such as Rational Rose, ARTS, etc., to be applied to the selected baseline project and by measuring the benefits obtained in terms of higher customer satisfaction,

reduced effort per requirement, reduced time to market, reduced rework and higher software quality.
An existing development team will employ such new techniques to the baseline project, comparing them against their own past experiences made with a traditional methodology (i.e., comparing the situation to the existing development process). During the experiment, the clients will be confronted with the results produced using the new method. As well, the outputs passed to the next phase of development will be evaluated for their added merits as input to the downstream development efforts. Finally, clients, users of the output deliverables and the development team members themselves will be surveyed by means of a questionnaire to determine the relative levels of satisfaction achieved with the new requirement specification process. The duration of the project is 18 months.
As the reader may know, PIE projects' tasks are usually splitted into Work Packages (WPs): the RESPECT workplan is divided into 8 workpackages: WP0 Project Management, WP1 Cooperation with other ESSI projects, WP2 Tools Acquisition and Integration, WP3 Predefinitions and experimental design, WP4 Training, WP5 Experimentation, WP6 Analysis and Consolidation of Results, WP7 Dissemination. A complete description of each single WP can be found in the RESPECT project proposal{footnote Also available on line on the project Web server at the Internet address:
http://www.ftcoop.it/RESPECT/HOME.htm}.

3 The Work Packages' activities

Although this paper is not intended to substitute project's deliverables, we want to highlight some of the experiences we made. Therefore in what follows we will shortly describe the main achievements reached within the already completed WPs.
In particular, we will present in Section 3.1 the tools selection activities of WP 2, also explaining the DESMET methodology and how it has been applied to this task. Section 3.2 and 3.3 will shortly summarize some of the difficulties tackled in introducing methodological guidelines for requirements definitions and in training activities.

3.1 Work Package 2: Tools Acquisition and Integration

There are several problems to be faced when a software development group in a company starts undertaking the evaluation of some new tool or methodology that could be later adopted. One of the most critical, and difficult to cope with, is that in addiction to technical difficulties, there are two other major concerns that need to be considered: the influence of human factors and sociological issues such as staff motivation and evaluator expectation, that can bias the evaluation exercise [11].
Although one may think that it is the common practice to keep the evaluation free from the influence of human factors, usually this is not the case. For example, if staff undertaking an evaluation believe a new tool is superior to their currently adopted tool, they are likely to get good results. However it may happen that the favorable results might not carry over to other software staff who do not have the same confidence with the tool. The DESMET project attempted to address this problem, providing guidelines to be followed in order to keep the evaluation as free as possible from the bias of human factors [9].

3.1.1 The DESMET methodology

One of the task to be performed within the RESPECT project was to screen the large variety of available tools, in order to select among all potential candidates the tool(s) to be adopted to support the requirements definition phase and the analysis and design phase. Of course, we had to face the problem to perform an objective choice, keeping the decision as free as possible from the bias of human factors.
The methodology we adopted to guide tools screening and selection is that suggested by the DESMET project [10], our reference for all the aspects concerning both qualitative and quantitative evaluation.
The DESMET project was a collaborative project partially funded by the U.K. Department of Trade and Industry, which aimed at developing and validating a method for evaluating software engineering methods and tools.
The DESMET method wants to identify which among the alternatives is best in specific circumstances: it supports evaluations aimed at establishing method/tool appropriateness i.e. how well a method/tool fits the needs and culture of an organization. Thus, the approach is context dependent (except with formal experiments): it is possible that an evaluation in one company would result in one method/tool being

identified as superior, but a similar evaluation in another company come to a different conclusion. We could say that the method support decision still leaving space to subjective choices.

Quantitative evaluation aiming at determining expected benefits from a tool and data collected to assess if such benefits are actually obtained will be performed in the experimentation work package. A qualitative tool evaluation carried out accordingly to what suggested by DESMET is based on a comparison among several alternative options, with the purpose of identifying which among the possible alternatives is best in the FTC specific context. The qualitative evaluation that we performed is also termed feature analysis. It is based on identifying a set of requirements considered important in company activity. These requirements are then mapped onto features that a tool should support. Each tool undergoes an investigation to assess the extent to which alternative tools support the identified features. The first step is defining a set of major and minor characteristics, that we will call tool features in what follows, that should hopefully be supported by a requirements engineering tool. As suggested by DESMET, these features will help select a set of tools that could be a potentially candidate for adoption by FTC. The identified features will represent the basis on which the feature analysis will be carried out on the set of previously identified tools, each of them providing facilities for the requirements phase of the software development process. Characteristics and features of each tool will be considered in order to select the best suited tool to support the FTC requirements phase, that will finally purchased and adopted by FTC.

3.1.2 Results of feature analysis

The tool evaluation following what suggested by the DESMET methodology was conducted through three sequential steps [10]. However it must be noticed that each step has to be iteratively repeated in order to refine collected information while the evaluation activity goes on.

Feature selection. A set of major and minor characteristics, i.e. tool features should hopefully be supported by the tools. As DESMET suggests, these features will help select the set of tools that could be potentially candidate for adoption.

The first step is about the definition of a feature list. Items in this list should match technical and performance desirable characteristics, as well as economical and compatibility issues. Each identified feature has then

been scored as suggested by DESMET: Mandatory (M), Higly desirable (HD), Desirable(D), and Nice to have (N).
The selected features are listed below (the associated scores are in brackets){footnote 2. Features are presented and not explained. In general they are self-explaining, and easy to understand. However the interested reader can find more details in the RESPECT Deliverable D 2.1 "Tools evaluation report", available on line at the RESPECT site: http://www.ftcoop.it/RESPECT/HOME.htm.}:

1. Software Requirement Specification (SRS in what follows) document related requirements

(a) (D) multiple SRS template
(b) (HD) functional and non-functional requirement template forms
(c) (M) traceability matrices and/or traceability links
(d) (D) keywords and keywords search facilities
(e) (HD) requirement ID coding convention and search (or other similar mechanism)
    i. per ID
    ii. per area
    iii. per category
(f) (D) status bar and search facilities by status
(g) (M) SRS documents repository and multiple version supports
(h) (N) traceability towards other process artifacts (e.g., project design, system testing)
(i) (M) UML support with special regards of Use Cases (see also 2.1)
(j) (M) requirements change impact analysis
(k) (D) HTML compatibility

2. UML related requirements:

(a) (M) Use Cases support
(b) (M) Java and Smalltalk support
    i. (M) class generation support
    ii. (HD) reverse engineering support
(c) (D) UML diagram subdivision into mandatory (e.g., class diagram) and optional (e.g., status diagram)
(d) (HD) template software project document support
(e) (M) interoperability with the existing FTC environment
(f) (D) reuse support

3. UML and SRS common requirements:

(a) (HD) in line help
(b) (HD) in line documentation and manuals
(c) (HD) help, tool documentation and manuals in Italian language
(d) (HD) support for the Italian language
(e) (M) training course in Italy
(f) (D) on-site technical support
(g) (HD) technical support in Italy
(h) (M) compatibility with international standard ISO, IEEE, CMM
(i) (HD) work-group and cooperative works support
   i. roles
   ii. security
   iii. protection

4. Economic issues:

(a) (M) purchase cost (training excluded) within approved budget

5. Supplier:

(a) (D) Well known company
(b) (HD) Represented in Italy

6. System requirements:

(a) (M) Compatible with Windows 95
(b) (HD) Compatible with Windows NT server

Although many different features set could be chosen, the iterative approach supported by DESMET helped us to refine the list until it finally made us confident that items presented above covers all the main characteristic for a tool to support conceivable FTC requirements, analysis and design phases.
Tools screening. The identified features will represent the basis on which the feature analysis will be carried out on the set of previously identified tools.
Technical reports, tools advertising, as well as searching the Internet {footnote 3 We have to mention an Internet site that provided a survey

about requirements tools, supporting useful indications and links: http://www.incose.org/workgrps/tools/tooltax.html.} and reading computer magazines provided initial information about the tools.
Tool selection. Tools vendors have been contacted: they provided additional information about products and an evaluation copy to install. After some bubbling, we improved on the field our knowledge about the tools, becoming able to exploit their features. At this point, we scored each feature for all the available tools, refining our judgment on a systematic basis.
Once that characteristic and features of each tool had been carefully considered in order to select the best suited tool, we were much more confident about the choice. Finally, Rational RequisitePro and IBM Visual Age were adopted and purchased by FTC. Here we recall some consideration about the selected tools.
Rational RequisitePro. This tool is based on a widely known editor - Microsoft Word, and is tailored to produce requirements documents exploiting a suite of available templates or by newly user defined one. Microsoft Word act as the editor / front end to a mechanism which interfaces by means of the Object Link Embedding (OLE) word documents with an Microsoft Access standard database. It keeps aligned the produced requirement documents and the associated database, performs key based document search and maintain traceability. Requirement documentation is compliant with international standards such as IEEE 830-1993. RequisitePro adopted technologies allow an easy integration with FTC already deployed software requirement documents and more generally with the FTC tool suite. The tool's interface is friendly and, after a short bubbling to get familiar with the command menus, the usage is quite intuitive.
IBM Visual Age. VisualAge is an IBM family of tools for the integrated design and development available for many languages such as COBOL, Java, C++ and Smalltalk. The tool point of strength is its strong support to the SmallTalk language integrated with a plug-in UML centered design module. UML diagrams are not completely supported, but the available diagram (class diagram, Use Cases and sequence diagram) will suffice for FTC purposes; all the supported views are well integrated and it is easy to navigate from one view to another down to the code.
It is important to highlight that this tool offers useful features for group-work: rules and roles are exactly the same used by the SmallTalk environment, thus allowing users and system administrator or project leader to play the same role in the design phase too. This is made possible

by the fact that Visual Age and SmallTalk share the same repository. Another interesting point is the tool's capability to generate HTML documents organizing information about the model under development. FTC development framework is undergoing the porting its Park Place Visual Smalltalk Enterprise software under IBM Visual Age Smalltalk, by a FTC software provider. This activity started in the early 1998 will be completed by the end of the year; since 1999 FTC will be forced to adopt the new framework. Hence Visual Age and its plug-in seem extremely appealing from FTC point of views.

3.2 Work Package 3: Predefinitions and experimental design

The introduction of requirements' definition and analysis support tools enriched the development environment, teaching by itself a more structured approach to programming activities.
However the definition of guidelines for the requirements phase provide a way to better structure the development process, making the FTC software development team evolve towards a more rigorous and formal process. Results obtained within the RESPECT project can be seen as a first step, that will be probably followed by a step to step definition of other software development activities.
The proposed guidelines have been inspired by those proposed by IEEE Computer Society [1]. However a systematic work was needed in order to adapt (and sometimes simplify) general indications from international standards to the FTC development environment, taking advantage from the available tools and experience.
For example, the previous practice was collecting user expectations by means of a Microsoft Word text document. For this reason the proposed guidelines suggest to start collecting requirements writing an informal "Interview Document". Once user's expectations are better understood, this document must be refined into a "Requirements Document". Advantages of the suggested practice are twofold: the previous experience finds natural reference into the new practice, and the suggested steps are directly supported by the selected tool (Rational RequisitePro).

3.3 Work Package 4: Training

Training activities about software engineering fundamentals provided a way to make people of the software development team enrich their background and making them growing as professionals. However, unlike

academic courses, lessons had to be targeted for skilled practitioners, who never want to hear about abstract concepts. For this reason, lessons were on the field: always addressing practical issues, and trying to answer real word questions.
Unfortunately, since software engineering is still a young discipline, in many practical cases there are no widely accepted solutions to be teached. People from ITC-Irst charged of training activities had to carefully provide a view of problems, balancing pros and cons of different approaches.
On the other hand, this more balanced way to present problems and approaches to their available solutions needed to be supported by practical examples. One of the major concern was not to generate the feeling that theached arguments are simply academic stuff.

4 Conclusions and future actions

The ESSI Process Improvement Experiment RESPECT has given the chance to start enhancing the software development process, making a small organization evolve from a unstructured and ad hoc process to a more formal one.
In many practical cases, software engineers and developers tend to restrict their interests to what they daily use: the RESPECT project gave an interesting opportunity to make involved people growing as professionals, improving their knowledge about software engineering fundamentals as a basis for project's main steps.
The first six months of work taught several interesting lessons:

Applying the DESMET method we organized the feature analysis and tool selection in a rigorous way, keeping the choice as free as possible from sociological issues and human factors.

Training activities about software engineering fundamentals provided a way to make people of the software development team growing as professionals. Unlike academic courses, lessons were on the field: they had to start from practical issues, thus answering real word questions.

The definition of guidelines for the requirements phase provide a way to structure the development process, making the FTC software development team evolve towards a more rigorous and formal process. Results obtained within the RESPECT project can be seen as a first step, that will be probably followed by a step to step definition of other software development activities.

The introduction of requirements' definition and analysis support tools enriched the development environment, teaching by itself a more structured approach to programming activities.

Although the project is still ongoing and we do not have quantitative results about requirements phase effort nor requirements stability, we feel confident that many useful qualitative lessons have been already learned.

References

[1] IEEE Guide to Software Requirements Specification. IEEE CS Press, New York, 1984.

[2] G. Antoniol, F. Calzolari, L. Cristoforetti, R. Fiutem, and G. Caldiera. Adapting function points to object oriented information systems. In CAiSE*98. The 10th Conference on Advanced Information Systems Engineering, Pisa, Italy, June 1998.

[3] G. Antoniol, R. Fiutem, and L. Cristoforetti. Design pattern recovery in object-oriented software. Sixth Workshop on Program Comprehension, June 1998.

[4] G. Antoniol, R. Fiutem, and L. Cristoforetti. Using metrics to identify design patterns in object-oriented software. to appear in Proc. of the Fifth International Symposium on Software Metrics - METRICS98, 1998.

[5] N. Bevan. Quality in use: Meeting user needs for quality. In Proceedings of the 4th International Conference on Achieving Quality inSoftware, pages 89-96, Venice, Italy, March/April 1998.

[6] F. Calzolari, P. Tonella, and G. Antoniol. Dynamic model for maintenance and testing effort. In International Conference on Software Maintenance, pages 104-112, Bethesda, Maryland, USA, November 1998. IEEE Computer Society Press.

[7] F. Calzolari, P. Tonella, and G. Antoniol. Modeling Maintenance Effort by means of Dynamic Systems. In Proceedings of the 2nd

EUROMICRO Working Conference on Software Maintenance and Reengineering, pages 150--156, Florence, Italy, March 1998.

[8] e. J. McDermid. Software Engineer's Reference Book. Butterworth-Heinemann, Linacre House, Jordan Hill, Oxford OX2 8DP, 1994.

[9] B. Kitchenham. A methodology for evaluating software engineering methods and tools. In Experimental Software Engineering Issues: Critical Assessment and Future Directions. Internation Workshop Proceedings, pages 121--4, Dagstuhl Castle, Germany, 14--18 Sept. 1992. Germany Springer-Verlag Berlin, Bermany 1993.

[10] B. Kitchenham. Desmet: A method for evaluating software engineering methods and tools. Technical report, Department of Computer Science, University of Keele, U.K., August 1996.

[11] B. A. Kitchenham. Evaluating software engineering methods and tool. 3. selecting an appropriate evaluation method - practical issues. SIGSOFT Software Engineering Notes, 21(4):9-12, July 1996.

[12] I. Sommerville and P. Sawyer. Requirements Engineering. John Wiley and Sons, 1997.

[13] P. Tonella, G. Antoniol, R. Fiutem, and E. Merlo. Flow insensitive C++ pointers and polymorphism analysis and its application to slicing. Proc. of the Int. Conf. On Software Engineering, pages 433-443, 1997.

[14] P. Tonella, G. Antoniol, R. Fiutem, and E. Merlo. Points-to analysis for program understanding. Proc. of the International Workshop on Program Comprehension, 1997.

SESSION 10:
Industrial Experience Reports

Establishing SPI Effect Measurements

Jakob H. Iversen (iversen@cs.auc.dk)
Department of Computer Science, Aalborg University
Aalborg, Denmark

Abstract

A problem in many software process improvement projects is how to determine the effect that the improvement initiative has resulted in. This paper reports from an organisation that has attempted to implement a measurement programme with the expressed purpose of measuring the results of the improvement project in quantitative terms. The company has attempted to establish an effect measurement programme that measures key indicators of all completed projects, and summarises results in a quarterly report. Based on the description of the measurement programme, some practical advices are presented for other companies who wish to establish similar programmes.

Introduction

Software Process Improvement (SPI) has become one of the major change strategies for software developing companies in the 90s. Maturity models such as CMM, BOOTSTRAP, SPICE etc. have been the traditional focus of the SPI efforts in many companies. Common to the many general approaches is a focus on establishing sound project management practices before attempting to implement more advanced and organisation-wide techniques (Grady, 1997; Humphrey, 1989).

A weakness in most SPI efforts, however, is the lack of focus on measuring the effects of the improvement in business and other terms independently of the maturity models. With no data on how the different improvements work, it is impossible to say if it has been worthwhile to implement the improvements or not. Although the literature on this subject is not extensive, some studies have been published, as referenced in (Emam and Briand, 1997);

Empirical studies that have been conducted do not answer all of the questions about SPI; those that have been answered not to the level of detail that some may wish. However, the available data do provide us with credible guidance in our SPI efforts, which is undoubtedly preferable to no guidance.

By measuring concrete phenomena and thereby obtain global numbers for the entire company, it is possible to get a detailed and accurate picture of the state of the software development work. It is then possible to decide whether initiated improvement activities have had any effect on factors that are important in relation to the company's business goals. The next section of the paper presents the case organisation, its SPI project, the approach used to obtain information about the organisation, and a historical perspective on the effect measurement programme. Section 3 contains a number of lessons learned, and the paper is concluded in section 0.

Case: Financial Software Solutions

Financial Software Solutions[32] (FSS) is a subsidiary of Financial Group. Financial Group provides all aspects of financial services (banking, mortgaging, insurance, etc.). The primary business function of FSS is the development of IT systems for Financial Group, but FSS also sells IT systems to other financial institutions across Europe. FSS has expertise in the development of banking, insurance, mortgage and financing applications. FSS has approximately 850 employees located at four geographically dispersed development centres.

FSS was recently made into an independent subsidiary delivering IT services to Financial Group. In principle then, Financial Group would be free to choose other providers, if they were dissatisfied with FSS. Faced with this reality, it has become a major thrust for FSS to stay competitive. SPI is seen as one of the strategies for keeping the business with the rest of the corporation. One of the most significant changes that occurred when FSS was established as an independent company was that all development work should be conducted in projects governed by contracts. This has even been extended to internal projects, which are now also under contracts with their internal customers.

[32] The name of the company and all actors has been changed to keep them anonymous.

The SPI project was given a very high profile in the organisation by appointing an experienced Vice President as project manager, and let other Vice Presidents be the project team members. When a project is normally managed by a Vice Presidents it is considered very high profile and important to the business of FSS. Further adding to the image of an extremely high-profile, organisation-wide project, the SPI project refers directly to a steering committee consisting of all the senior managers (Chief Executive Officer and four Senior Vice Presidents). The SPI project organisation is shown in Figure 1.

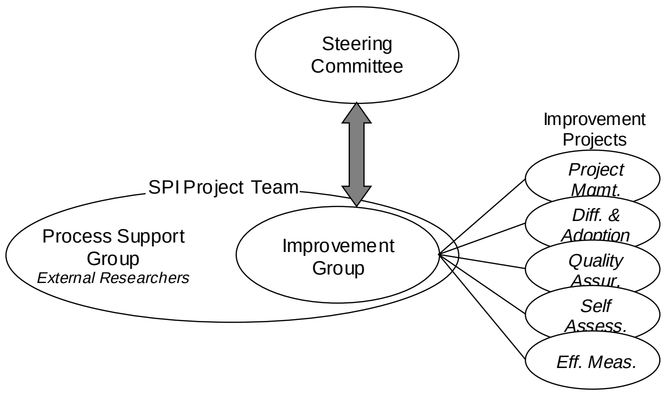


Figure1 Organisation of the SPI project in Financial Software Solutions.

The improvement group acts as a catalyst for the actual improvement effort. Improvement projects, each of which deals with one improvement area, make the detailed decisions on how and what to improve. The improvement projects are staffed with people that are knowledgeable in the area that they are going to improve and well respected in the organisation. The researchers have also involved themselves in these improvement projects, and are thus able to provide the FSS members of the groups with additional insights and inspiration in return for the added insight and understanding of SPI that being involved in such groups give to the researchers. Currently, the following improvement projects are ongoing: Project management (education and Project Manager Self Assessment), Diffusion and adoption of methods and techniques, Quality assurance in projects, and Effect Measurement.

Research Approach

This research is part of a large research project involving four software-developing companies, two universities, and a consultancy company. The researchers and consultants participate actively in the SPI projects of each of the four companies over a three-year period. The SPI project in FSS was initiated along with the research project in January 1997. The research project will end in December 1999, but it is expected that the SPI project will continue beyond that.

At FSS, the researchers and consultants (commonly referred to as 'researchers') are active participants in the improvement group, and the research methodology applied was thus action research (Foster, 1972). The main interaction between the researchers and the organisation took place at the monthly SPI meetings, but also by more informal meetings, working sessions, workshops etc. in which only a single improvement initiative (in this case effect measurement) was discussed. As the researchers became part of the SPI organisation they were able to obtain real insight into what the issues facing the SPI project were.

Two of the major problems in conducting action research is 1) the limited ability to generalize findings (Mathiassen, 1997), and 2) the frequent neglect by many action researchers to collect adequate data to be able to demonstrate convincingly what was learned during the study. The former problem is dealt with in this paper by determining a number of practical lessons that were learned both by the researchers and the internal SPI members. These lessons are conveyed as practical advice to other companies involved in establishing similar problems. In this research we have attempted to overcome the latter problem by systematically collecting as much data as possible about the organisations. This included all the 'natural traces' of the SPI program such as project plans, meeting minutes, memos etc. In addition to this, we have tape-recorded the monthly SPI meetings as well as some of the working sessions and workshops. The relevant segments for effect measurement were transcribed.

SPI Effect Measurement

The focus on measurements is specifically intended to enable the SPI project and senior management to make informed decisions about the improvement activities as well as to assess the effect and progress of these activities. Apart from giving guidance to the improvement group and the improvement projects, the measurements are also seen as a way of getting some attention from the rest of the organization on the SPI project.

Table 1. Key stakeholders of the effect measurement programme in Financial Software Solutions

Chief Executive Officer (CEO)	Sponsor of the SPI project. Stated that FSS should improve efficiency by 10% through SPI.
Vice Presidents	Responsible for 20-25 people and 3-5 projects, they are what the CMM terms a "first-line software manager"., and are thus crucial in implementing suggested improvements.
Project Managers	Required to report data on their project in the central time and project registration system (TIRE/POKA), and for providing the effect measurement project team with a few central pieces of information about the project.
John	Project manager for the SPI project.
Ashley	Full time employed on the SPI project. Project manager for the effect measurement project.
Finley	Vice President and member of the SPI improvement group. Was heavily involved in defining the first measurement programme.
Linda	Vice President and member of the SPI improvement group. Has not been directly involved in the effect measurement programme.

Table 2. Timeline of key events.

Year	**1997**												**1998**												**1999**											
Month	J	F	M	A	M	J	J	A	S	O	N	D	J	F	M	A	M	J	J	A	S	O	N	D	J	F	M	A	M	J	J	A	S	O	N	D
Event	1		2						3						4 5					6								7 8								

This section presents the process of establishing the measurement programme in Financial Software Solutions. The presentation is structured around eight events that have influenced the establishment process. The description of each event in some cases includes what happened immediately prior to the event and what followed immediately after it. Some of the key stakeholders are listed and described in Table 1, and Table 2 shows a timeline of the individual events.

Event #1: Press Conference. January 7, 1997

One of the first things that happened in the overall SPI project was a press conference. At this conference, the CEO of FSS stated that

> *"we expect to gain a 10% improvement in efficiency through this project … that is equal to 30 mill. DKK."*

This statement became the focal point of the SPI project in FSS. In the SPI project contract, one of two critical success factors of the SPI project were

> *"that FSS within the project's 3-year time span has achieved an improved efficiency of the system development process of at least 10%."*

From the beginning of the project it has thus been very important to be able to show this 10% improvement. However, neither the CEO nor the contract were explicit on what should be measured and how the data should be analysed to show the improvement. This was left to those members of the SPI team that were made responsible for implementing the effect measurement programme.

Event #2: Decision to Implement Effect Measurements. March 1997

After some input from the researchers, the improvement group decided to establish an effect measurement programme, to measure the 6 factors listed in Table 3.

Table 3: Indicators of the measurement programme in FSS.

Factor	*Definition*
Project Productivity	Resources used to develop the system relative to size of project in Function Points
Quality	Number of error reports both absolute and relative to size of project in Function Points
Adherence to schedule	Variation from agreed time of delivery both absolute and relative to size of project in Function Points
Adherence to budget	Variation from estimated use of resources
Customer satisfaction	Satisfaction with the development process and the implemented solution (multiple choice questionnaire)
Employee satisfaction	Satisfaction with the development process (multiple choice questionnaire)

A decision memo from May 13, laid down some of the principles that the effect measurements would adhere to:

- *Measurements should be relevant in relation to process improvement and quality, but also have general management interest.*

- *Measurements should as far as possible be made automatic. Information should be interpreted rather than disturb the development organisation.*
- *Cost of performing the measurements should be minimal.*
- *Use of questionnaires should be limited as much as possible, as the organisation suffers from 'questionnaire-disgust'. If questionnaires are used, they should be placed at a milestone, and be adapted to the natural system development process.*

Data should be collected on projects that were finished (to keep disturbances to a minimum) and results should be published every quarter. The volume of the projects should be calculated using an automatic counting algorithm for Function Points (IFPUG, 1994). Function Points (FP) are usually calculated by experts with a significant amount of practice in counting FP. Not many organisations have attempted counting FP automatically, and it was therefore risky to engage in this work. Therefore, the project concentrated some effort (app. 1-2 man months) on implementing the automatic calculation system.

Event #3: First Measurement Report. September 1997

The first visible result of the measurement programme was the first measurement report, completed in September 1997 with results from 13 of 56 projects that were completed in 3Q97. The report had data on 3 of the 6 factors (adherence to budget, time-to-market, and project productivity). The data contained some surprising information especially regarding adherence to budget causing senior management not to make the report public. Parts of the results were instead disseminated to the development organisation through a 'roadshow' conducted by the improvement group to raise awareness towards the SPI project. The report was also criticised for being too academic. A workshop was held in October 1997 to improve the layout of the report to alleviate this problem. The problems in gaining acceptance for this first report did not bother the improvement group significantly, as it was, after all, the first report, and was seen as something of an experiment with less than optimal data foundation.

Event #4: Second Measurement Report. March 1998

Data for the second report, covering projects that completed in 4Q97, were collected in October through December 1997, and the report was completed in March 1998. The results were discussed in an SPI Project Team meeting on

February 20, 1998. Data discipline had been greatly improved as shown in Table 4, although there ought to be 100% complete data sets.

Table 4. Data discipline in the effect measurement programme.

		Complete data sets	
Period	# projects	#	%
3Q97	56	21	37
4Q97	29	19	65

This improvement in data discipline and thus in data quality was received with much enthusiasm at the meeting. However, the discussion soon centred on the issue of whether the report should be made public or not. At the meeting, there was some disagreement on how to distribute the report and how detailed the information should be:

John: "It could become publicly available. If we publish the main figures, and then anybody could request a copy. What will happen in the report is that projects will be mentioned with their names. In the first report, they were anonymous. They aren't sissies in the senior management group."

Linda: "I don't think it should just be a matter of requesting a copy. [The report] should be spread."

Ashley: "I'm … a little nervous about including names and so on."

John and Linda had not been directly involved in the effect measurement programme. Their interest is to improve the SPI project's visibility in the organisation. Ashley, however, had personally promised many members of the development organisation that they would not be personally blamed for any bad results, so she is naturally not interested in getting personal information spread too widely. The researchers tried to help by suggesting alternative solutions:

Lars (researcher): "what is actually unpleasant today is worst-case: there are measurements of people, and they know they are there, but they don't know what the measurements are. […] We can only win by getting these measurements out. There is also a solution that what is published is per department, so that those in department 2 can't see department 3. [...] But they need to get some concrete information back."

Jan (researcher): "I think it could give an unfortunate effect to make the numbers too widely available, because [...] someone may try to make their numbers look better than they are. However, if they get an average plus their own project and then are encouraged to [discuss internally] why the numbers look the way they do. [...] I think that will give a good effect."

As it can be seen, there was no common understanding of what it would mean to make the numbers public. However, there was general consensus that senior management had accepted the idea that the measurements should be made public:

> *Linda: "I think we have [the CEO's] commitment that now he will [make it public], and we should of course make sure he sticks to that, once the report is completed."*

The report was finished in March 1998, and had a much clearer layout than the first report. The report did not include productivity data because the automatic calculation of Function Points was considered faulty. Instead, data on customer and employee satisfaction was included.

Event #5: Decision Not to Disseminate Second Report. March 31, 1998

The report was presented at the steering committee meeting on March 31, 1998. The data was considered insufficiently reliable to warrant a wide distribution of the report. The results of the satisfaction surveys showed very unfavourable results for key business areas for FSS, and the steering committee thus decided to withhold the second report also. The presentation of the report to the steering committee was discussed at the SPI project team meeting on April 22, 1998:

> *Meeting minutes: "The report is not and will not be made public internally in FSS. A strong contributing factor to this is that the credibility of the data validity is insufficient."*
>
> *John: "When I presented the report, I did so from a positive angle throughout. Emphasised what was good, and the positive things that had happened since the last [report]. Data discipline has improved. … Some of the data discipline is due to the information meetings. People have become aware that this is actually used for something. Of course, it can still get better."*

One of the major problems in the report was the low rating of how FSS supported the business function of Financial Group. However, it was quickly identified that the customer satisfaction questionnaire was of poor quality and primarily directed towards customers, whereas in most cases users who had not been involved in negotiating terms and contracts, had answered it:

> *Finley: "We send these questionnaires [...] to people who were involved in the acceptance test of the system. And what we then ask are managerial, contractual, overall process-related questions on how the project was conducted. Then some random user has to answer if commitments were met. He hasn't seen the contract or*

anything. It’s bound to go wrong, and that's why management can’t recognise reality in these numbers."

The issue of management commitment towards effect measurements was discussed as well:

Finley: "I don't know how much management commitment we have here. This is the second report. We ask for more resources, but nothing much is happening. We must end up with something that gives management a credible picture of reality. Otherwise they will say: this measurement stuff - forget it, I'll be better off trusting my intuition about how the reality actually looks."

Linda: "I get personally disappointed that [the CEO] does not release the report. I can understand that he is afraid of [the bank's central IT co-ordinator], but if we are ever going to get people interested in SPI, then they need to see what the project is doing."

This event is probably the most important in the history of the effect measurement programme. It caused a dramatic increase in the attention given to the programme, and caused establishment of a project to improveme the effect measurement programme. This illustrates how difficult establishing such a programme is. A lot of resources had been used on defining each metric, and deciding how to measure them. But some aspects had still been overlooked: the questions in the satisfaction questionnaires had not been carefully formulated, and the customer questionnaire was given to users instead of customers. On top of that, insufficient attention had been given to incentives for the development projects in reporting the necessary data, resulting in poor data discipline.

Event #6: Improvement Project Established. August 1998

After the disappointment that the report was not made public, the discussion in the SPI project team meeting on April 22 focused on actions that could be taken to improve the effect measurement programme enough to enable publication of the next report. The group decided to try and establish a project to improve effect measurements. The project was proposed in June 1998, and the contract was signed in August 1998.

The project was established as an improvement project with the success criteria that a measurement report is completed in April 1999. This report should contain data on all 6 indicators and from all projects completed in 1Q99. Compared to the second report, this report should have improved the measurement process for all the indicators, and the layout of the report should also be enhanced. Among other things, the data quality of each indicator should be displayed, for instance as a percentage of complete data sets (as in Table). The main goal was that the quality of the measurement report should be improved so much that it would be impossible for management to deny making it public

Event #7: Third Measurement Report. Planned for April 1999

While the improvement project has been running, a bonus system has been introduced, in which one of several factors is the correct reporting of data for the effect measurement programme. The preliminary results of the improvement project indicate that the data discipline is still considered insufficient, as it has not been possible to persuade projects close to finishing that they should report data in the new format. However, it could be discussed whether this would actually make the data less valid, because one of the factors that is frequently missing, is an estimate of the expected date of system delivery. Reporting this figure when the project is almost complete would make for complete data, but with very little relevance. For projects that have started more recently, things look a lot more positive. This is in part also due the bonus program requiring projects to report accurate and correct data.

It was decided that FP would not be included due to the problems of making an accurate count. The possibility of using other size measures has been examined, and rejected as none of the measures proposed (lines of code, compiled size, function points, and number of statements) all had severe weaknesses that made it better not to include a size measure at all. Excluding a size measure seriously impedes reaching the original objective of measuring efficiency and productivity, as there then is no measure of the output of the project.

The satisfaction surveys have been integrated well into the quality assurance process. The projects give the questionnaires to their customers and employees and the results are discussed in quality meetings towards the end of the project.

Event #8: Report Disseminated Widely. Hoped for April 1999

The fate of the measurement programme will be decided after the first report has been completed. If the report is not made public within Financial Software

Solutions, then it is likely that the effect programme will collapse. Not having an effect measurement programme will cause serious problems for the rest of the SPI project, as many of the initiatives are driven by the effect measurements.

Lessons Learned

Despite having worked concentrated on establishing effect measurements for almost 2 years, it seems that nothing much has really happened. The two reports that have been prepared have not been distributed as widely as was desired and presently, the programme is put on hold until a major improvement of the concept has been carried out. It is not all negative, however. The organisation has gained significant experience in this area and some data has been collected that has contributed to significant discussions about the validity of previously held beliefs and intuitions about how the organisation operates. The experience that FSS had in establishing the effect measurement program has also given rise to a number of lessons, which may be valuable for other companies attempting to establish similar programmes.

Start Simple

The wish to measure six indicators, each of which was of a complex nature with no previous measurement process in place, can only be characterised as extremely ambitious. It is easy to become disappointed when the collected data doesn’t have the expected quality, and measuring some indicators must be abandoned. Another approach is to start out by simply measuring one or two indicators, perhaps just collecting data that is already there and just analysing it. Later, when the measurement team and the development organisation have gained experience in measuring and being measured, other measures could be added to improve the programme. Such a staged introduction of a measurement programme may take longer than the ambitious approach. But the results may also be less disappointing.

A Real Project

At first, the measurement programme was considered as an integrated part of the SPI project. In this phase, there was no plan for the work, and the objectives were described in vague terms in the SPI project contract. It was only because of the dedication of a few employees that a measurement programme was established at all. Later in the process, when a real project was established, it became far easier for the actors to argue that they should have adequate resources and the focus on the initiative was generally increased.

Establish Incentives

The FSS case clearly illustrates the value of establishing incentives to improve data quality. From the outset, all projects were required to record several numbers in the on-line project and hour registration system, but almost no projects recorded complete and correct information. Mainly because they saw no immediate use for the data they provided. A marked improvement of data quality was achieved by using a combination of informing the project managers of what data they should report and how to do it, as well as informing about the importance of the data they provided and show some results based on the data. However, when reporting accurate data became part of the bonus system, a very clear incentive scheme was established, and data quality now appears to be very good (for the projects starting after the bonus system was established).

Publish Widely

The biggest disappointment for the measurement team at FSS has been management's decisions to withhold the reports from distribution. In order to improve the measurement programme it is vital that the development organisation be given some feedback on measurements. Not making the reports public, can thus be seen as a barrier for improvement of the programme. However, it is also important that performance measures of individuals be kept to the individual. Otherwise everybody would do all they can to make their numbers look better, losing the entire purpose of the measurement programme.

Allow for Discussion

Establishing an effect measurement programme forms an important shift in culture towards a culture where decisions are based on factual numbers rather than vague intuitions. If the data quality is to reach a sufficiently high level of quality, the organization must be willing to make less than optimal data available for discussion about the validity of the data, and the assumptions underlying the measurements. But perhaps more importantly, it should be recognized that even if the validity of data is not as high as could be wished, the information contained in the data, might still carry some relevance. However, it is also important that the numbers not be taken as absolute truths. They are not, so it is vital that the data quality can also be discussed to improve the foundation of the measurement programme.

At FSS, this discussion has currently been limited to the SPI project team and mangement. Here, the discussion has been lively and inspiring to those participating. But as long as the discussion is contained to a small number of people, it is difficult to use the measurements as a feedback mechanism to the development organization to improve on the daily practices in the project.

Effect measurements are likely to give some very unpleasant results about the organisation. Being able to cope with such results and use them to improve the organisation rather than just figuring out who is to blame for the bad results is also part of this cultural change. Establishing such a culture is no small task, and, as the case description above illustrates, is bound to take a lot of time and effort on behalf of those who attempt to affect it.

Conclusion

Metrics in software development are usually primarily used to direct the course of a single development project (Basili and Weiss, 1984; Carleton, et al., 1992; Fenton and Pfleeger, 1997) and secondarily used for management overview of all projects. However, collecting data after a project is completed is far easier than while it is still running. At Financial Software Solutions this has been exploited to develop a measurement programme that uses post-mortem measurements to characterise the overall state of the state of the company's software development work. The information obtained from these measurements can thus be used to describe the effect of any improvement initiatives conducted in the organisation. The lessons learned from this experiment illustrate that it is far from a simple undertaking to establish such a program.

Whatever reasons management had for deciding not to make measurement results public within Financial Software Solutions, it certainly illustrates that they did take the results seriously and were concerned that releasing data that did not convey an accurate picture would be detrimental to the organisation.

Acknowledgement

Financial Software Solutions is thanked for granting access to their SPI project and providing excellent conditions for this research. The Danish National Centre for IT Research has financially supported the research. These colleagues have provided valuable input: Ivan Aaen, Jesper Arent, Gro Bjerknes, Karlheinz Kautz, Lars Mathiassen, Jacob Nørbjerg, and Rens Scheepers.

References

[9] Basili, V.R. and Weiss, D.M. "A Methodology for Collecting Valid Software Engineering Data," *IEEE Transaction on Software Engineering* (10:6), 1984, pp. 728-738.

[10] Carleton, A.D., Park, R.E., Goethert, W.B., Florac, W.A., Bailey, E.K. and Pfleeger, S.L. "Software Measurement for DoD Systems: Recommendations for Initial Core Measures," Technical Report SEI-92-TR-19, Software Engineering Institute, September 1992 1992.

[11] Emam, K.E. and Briand, L. "Costs and Benefits of Software Process Improvement," International Software Engineering Research Network technical report ISERN-97-12, Fraunhofer - Institute for Experimental Software Engineering, 1997.

[12] Fenton, N.E. and Pfleeger, S.L. *Software Metrics - A Rigorous and Practical Approach*, PWS Publishing Company, 1997.

[13] Foster, M. "An Introduction to the Theory and Practice of Action Research in Work Organizations," *Human Relations* (25:6), 1972, pp. 529-556.

[14] Grady, R.B. *Successful Software Process Improvement*, Prentice Hall PTR, Upper Saddle River, New Jersey, 1997.

[15] Humphrey, W.S. *Managing the Software Process*, Addison-Wesley, Pittsburgh, Pennsylvania, 1989.

[16] IFPUG "Function Point Counting Practices Manual Release 4.0," Manual The International Function Point Users Group (IFPUG), January 1994 1994.

[17] Mathiassen, L. "Reflective Systems Development," unpublished Dr. Techn., Aalborg University, 1997.

A PIE one year after: APPLY

Alain Roan (roan@verilog.fr), Patrick Hebrard (hebrard@verilog.fr)
CS VERILOG
Toulouse, France

Abstract

Keywords: PIE, ESSI, Configuration Management, Test, CMM, Requirement Management, Planning and Tracking Management, GQM, Software Engineering, Tools.

A Process Improvement Experiment was performed a year ago into a product-oriented organisation. This paper intends to show the longer effect of a PIE in this organisation. The main improvement goals were to achieve more reliable software with the same validation effort together with a better mastery of the time to market. Improvements were visible during the experiment and the dissemination was quite successful.

Summary

CS VERILOG is a manufacturer of software engineering tools. VERILOG was looking for an improvement of its internal practices to deliver better products to its customers and for an increase of its experience to help external organisations to plan and implement its technologies.

The APPLY (Amplified Process Performance LaYout) project was the experiment of the combined application of recommendations coming from a global Process Improvement Program initiated after a CMM self assessment. This assessment shown that ‘good heterogeneous’ practices were in place among technical groups (CMM level 1). One major weakness was found in the decision process about release content, availability date and reliability trade off. The diagnosis was very clear: define and apply a homogeneous process.

The main objective of APPLY was to implement the recommendations on a real size project in order to validate them and to obtain a quantitative project profile (cost break down and benefits) to facilitate internal and external replication.

APPLY results are:

- A better project control:
  - **Initial budget respected,**
  - **delivery time managed within a 5% window**
- A higher reliability of the final product:
  - **Remaining bugs are divided by 2 for the same test effort**

Economical consequences have to be evaluated on market side rather than on the internal productivity benefits. For instance, VERILOG has had 40% product sales growth and an increase of maintenance contract renewal.

Chapters 2, 3 describe the APPLY project. Chapters 4, 5, 6 give an analysis and a conclusion one year after its completion. This work has been financed by the European Commission as part of the ESSI program (APPLY 21.511).

APPLY project

1.1 Objectives

Three main constraints exist on every project in the organisation:

- Budget.
- Time to market.
- Quality of the release.

In the past, it was more frequent to miss one of these constraints than to satisfy all of them at the same time. The most crucial business objectives at this stage are: Reliability of the released product and Time to Market management.

Reliability

Time to Market

VERILOG is leading two types of projects:

- Projects founded by a customer. In many cases the customer tries to impose its own methods for development, management or quality assurance. This is of course not satisfactory from the point of view of a common process: it creates project specific procedures and methods. A consistent and well-documented process can be extremely useful to convince customers that practices in place are adequate and sufficient.

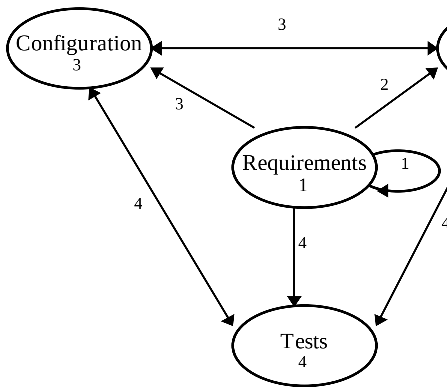
- Self founded projects where VERILOG itself and the market is the ’customer’. There is a natural tendency to be less rigorous on process and procedures application in this case. A group led by the Product Marketing Manager drives each product development. The main characteristics of such projects are: **Requirements are always changing until the end, the expected reliability level is high, time to market is key**

Key areas of improvement were:

- Management of Requirement/ Time to Market trade off:
    1) Better mastery of estimating and planning of tasks and project according to Requirements.
    2) Configuration management linked to planning and Requirements in order to be able to assess quickly requirement evolutions.
- Efficient tests and non-regression tests (time, effort and coverage) reusable for different target platforms after porting. A better traceability between Requirements and tests was also expected (validation).

1.2 Scope of APPLY

The improvement covered by APPLY were:



Efforts have been spent in sequence 1, 2, 3 and 4. The figure shows the dependencies between the four areas. Impacts of a requirement change for instance have to be evaluated on the three other areas.

1) Formal management procedure for Requirements in place. Whatever the way of expressing precise requirements (OMT and free text), requirement items are consistently managed. The tool ARS is used to managed evolution requests.
2) New planning rules and tools in place allowing to put formal planning impact analysis into the decision loop of Requirement evolution. MS Project is used and the internal tracking tool is linked to it.
3) New configuration management procedures and tools in place. Different elements of the project are automatically managed in a consistent way (work items, software code in different languages, requirements, tests...).
4) Formal test and validation procedures and tools are used to automate test capture, play back, and coverage measure. Formal links are established between Requirements and tests. The tools Logiscope, Purify and ValidX are used.

Area	Tools used
Planning and Tracking	MS Project, internal tracking tool
Requirement Management	MS Excel, Remedy/ARS, VERILOG/ObjectPartner
Configuration Management	Freeware/CVS + Parameterisation
Test & Validation	Softix/ValidX, Pure Atria/Purify, Numega/ BoundChecker, VERILOG/Logiscope

1.3 Project Control Panel

The first phase of the APPLY project consisted in the definition of a Project Control Panel. The goal of this Project Control Panel was to monitor the baseline project performance as well as the experiment itself in respect to the three selected business goals: **make time to market predictable, warranty a level of reliability, monitor improvements of the PIE itself**. Metrics have been defined using Goal/Question/Metrics approach.

The Project Control Panel was more difficult and longer to define than originally anticipated. The biggest difficulty is to have to wait actual data in order to be able to validate the original design. The Project Control Panel has evolved during the experiment in the direction of simpler metrics (see chapter 3 for a global view on the GQM tree).

One can argue that the business objectives are not very ambitious. We strongly believe that even if the real targets were higher, it would be unrealistic and misleading to proceed in a different manner. Before deciding to 'improve' or 'reach a given level', basic questions must be answered first:

- Do you know your present average quantitative software development performance?
- Is your performance reproducible from a project to another?
- Are you able to evaluate consequences of project decisions on performance?

1.4 Implementation of the new processes

APPLY was lead by Quality Department. A work package leader was nominated for each key area (Requirements, Planning, Test and Configuration). These people were not directly involved in the baseline project. Each of them is recognised by the baseline project team, as the most knowledgeable in their field.

The progress of the technical work packages was checked with generic milestones:

- Procedure Draft released
- Training & Experimentation started
- Controls of Application Started
- Measurable Metrics
- Final Assessment done

The approaches followed by work package leaders were either top down (define everything and then apply) or bottom up (try and formalise).

Results and Analysis

1.5 Global Achievements

The following table is an abstraction of the Project Control Panel at the end of the experimentation. Under each sub goal several questions were established and associated metrics were defined according to the GQM method. For instance sub goal G12 had three questions: Do we manage every files in the new system?, Do we have an automatic rebuild of the binary code?, Do we have an automatic rebuild of the final medium?. Some of the final metrics definition and results are described in the rest of the section.

The general objectives and final status indicated in the table are abstracted from the low-level objectives and achievements that have been fixed at terminal metrics level.

Business Goal	Initial High level Objective	FINAL
G1 – Make Time to Market Predictable		
G11 – Manage Software Baselines	100% of files are managed in the new system	OK 95%
G12 – Manage the Baselines Consistency	100% Traceability of Requirements into planning and test.	OK 100%
G13 – Maintain a Time to Market Best Estimate	predict release time within a 10% window	OK (vs. Latest estimate) NOK (vs. Initial)
G2 – Warranty a level of Reliability to the release		
G21 – Validate the release	80% of tests are automated 100% of tests have been run	OK > 80% on average NOK on two modules OK 100%
G22 – Monitor the Reliability Achieved	less than 10 known bugs in the release	OK (5) on average NOK on one module

G0 – Monitor Improvements of the PIE itself		
G01 – Monitor the progresses of APPLY	milestones in time (10% window)	OK
G02 – Ensure that the baseline project applies new processes	100% of compliance to new processes	OK 81% but increasing
G03 – Measure the cost of APPLY	in the initial budget (10% window)	OK
G04 – Measure the benefit of APPLY	see other goals	OK

1.6 Process Conformance

It is necessary to check that the new practices are in place. Otherwise no conclusion is possible. The measure of process conformance was made by a questionnaire (deduced from the written procedures). The percentage is the number of correct answers divided by the number of questions.

% of compliance	Feb 97	May 97	Nov 97
Planning	40	60	100
Requirement	20	40	60
Configuration	40	80	80
Test	78	78	83
Global	**51**	**61**	**81**

1.7 Requirements

Among the various data that has been captured about requirement management, three of them have been more particularly analysed:

Metrics	**May 97**	**Nov 97**
%approved req. items (initial + added) which are traceable into planning	84%	85%
Traceable implemented items/approved requirements items (initial + added)	89%	90%
Approved changes/proposed changes	30%	25%

The granule of requirement items was very heterogeneous in terms of amount of work to implement and verify (from one hour to months). This type of metrics was new in the organisation and we lacked previous measurements in order to make quantitative comparisons. But we can state that the values of the first two are probably much larger than before. This is the consequence of a better adherence to the process for the first two. The third one shows that 75% of proposed changes during development have been rejected. The reasons have not been analysed in details, but it is probably the consequence of a better visibility upon the impact of decisions.

1.8 Planning & Tracking

Metrics on the Planning & Tracking area were based on two sources of information. The first one is coming from the accounting department and gives budget and time envelopes allocated to the project. The second source is the planning and tracking databases that indicate the actual planned effort and forecast for termination.

Date	**-10% +5%**	**# budg. rev.**	**#plan rev.**	**Effort**	**Time**
96Q2	OK	0	1	96,1%	99,6%
96Q3	OK	0	1	97,6%	99,6%
96Q4	OK	1	1	97,8%	105,6%
97Q1	OK	0	2	101,0%	105,6%
97Q2	OK	1	2	105,6%	116,1%
97Q3	OK	0	3	97,7%	121,5%

The planning procedure states that when the planned cost of the project goes outside a window (-10%, +5%), the allocated budget must be officially revised (or the planning revised). The first column shows that this aspect of the procedure has been respected in the period. The two next columns are self-explanatory. The last two columns are the comparison between the initial effort and time and the planning forecast (whatever the approved changes in the budget). It is not a surprise to see that the effort and time are growing around the end of the project. Thanks to this visibility, the project allocated effort has been revised to recover the initial envelope.

The management of time is now integrated in the procedure: a planned delay of more than 30 days implies the revision of the project envelop. Despite this, the planned/initial ratio is still high (21%). This is not a bad performance in itself. It should be analysed in parallel with the requirement changes that have been made and accepted. In any case, the reliability of initial planning must be still improved.

1.9 Configuration

The achievements in the field of Configuration Management were captured with three progressive metrics:

Metrics	**May 96**	**June 96**	**May 97**	**Nov 97**
% of files managed in the new system	0%	18%	80%	95%
% of executable files that can be automatically rebuild	100%	?	100%	100%
% of files on the product medium that can be automatically rebuild	?	2%	27%	61%

The initial objective (100% of files managed in the new system) was almost reached (95%) but the goal became a full automatic procedure to rebuild the products.

At the end of the day, most of the practitioners have experienced a significant benefit in their daily configuration management practices . Only one group of people, which was the most advanced in the field, has lost some capabilities. Despite the global benefit of homogeneous practices they still regret the previous process. It must be used in the near future as an incentive to go one step further.

1.10 Test & Validation

The validation and test processes provides a lot of data (functional and structural coverage, bugs detected during validation). We have focused ourselves on two major aspects.

Proportion of tested requirements by test categories:

- automated test proportion has doubled (mainly with ValidX capture and replay).
- reproducible tests (fully documented context, inputs and expected outputs) represent now nearly 70% of the test base. The rest is still demanding a good knowledge of the product and its specification.
- 95% of the tests have been passed during the last campaign.

Evolution of remaining bugs (level 1 and 2) in time: Globally speaking, the number of remaining bugs in the release has been divided by two. But looking more closely to the individual modules reveals that the initial objective (less than 10 known bugs) is not reached on 2 out of 7 modules. Analysis of possible causes has revealed that the choices among the defects to be fixed have taken into account not only the defect severity (as the metric does) but also its frequency and impact on users. In other words, the criteria for deciding to fix a bug in a release are more complex than the bug severity. An improvement of the classification scheme was decided.

One year after

1.11 Dissemination

The status of dissemination is shown in the table. The foreseen future at the time of the project end has been kept unchanged for the sake of comparison:

Area	**Status (Nov 97)**	**Foreseen Future (Dec 97)**	**Status (Dec 98)**
Req.	Only the baseline project is using.	Generalisation is planned for all products.	OK
Plan.	Every project uses it in the main site.	Generalisation to the whole organisation	OK
Conf.	Half the projects plan to use.	Every new project in 1998 will use the new tool. All Verilog is products used APPLY method as the following: • Archiving technique only for every old product. • Full use of the Configuration Plan for the others.	OK
Test	2 other projects use or plan to use	More studies and partial generalisation	OK

Globally speaking we achieved 100% of the expected dissemination one year after. To be more precise two procedures have been generalised as they were at the end of APPLY (with minor modifications): Planning & Tracking practices, procedures and tools are homogeneous in the organisation for every projects as well as configuration management for every new developments.

Two procedures have been largely simplified but are now in place in the whole company: Requirement Management and Test

A new organisation for the Technical Department (around 40 people) has been put in place at the end of 1998. This is a matrix organisation with products in rows and types of software in columns. There are two dozens projects running simultaneously (development, maintenance and research). A group for Best Practices, in charge of product procedures and validation, covers every product. This should accelerate the initial momentum brought by APPLY in direction of even better practices.

1.12 Business

The product sales revenue growth (which is a major business goal) has been +40% in 96/97 and +30% in 97/98. On four consecutive releases of the same product the overall improvement in terms of project control capability and reliability have been dramatically improved. It is, of course, rather difficult to link these results to the improvement experiment only.

Anyway, for a business perspective, it is indubitable that benefits are very high:

- Direct financial benefit. **Better forecast and control of the initial budget**.
- **Better control of the release content**. A better competitive position is expected.
- Customers have **a better opinion of the company and its products.**
- The development team has a **better capability to anticipate** consequences of requirement changes.

1.13 Culture

The technical staff in place in the organisation is very skilled and experienced in the field of software development. Most of them are exposed and participate to advanced research in the field and have tried by themselves several up to date technologies. It is a chance and a handicap at the same time. On one hand, the introduction of new techniques is not a problem. On the other hand, creative people are not very keen to the idea of applying a common set of processes and tools.

When introducing uniform methods into an organisation with heterogeneous practices, classical resistance is encountered: individuals are naturally inclined to compare the advantages of the previous practices (the ones that they created) with the weaknesses of the proposed new ones. Generally the new ones are not so well suited to the particular context and are perceived as more rigid.

On the other hand, some reactions propose to go beyond the proposed practices. This is of course a very powerful force that has to be used for future improvements.

The challenge today is still to make understood and well accepted that homogeneous practices are a necessary milestone to further improvements.

It should be noted that quantitative measurements had a great impact on the improvement process itself. About the Configuration Management for instance, it is clear that the promotion of metrics results has originated its own improvement. This is true also for metrics for which no precise objectives were fixed (automatic rebuild of the medium for instance). In some cases, the metrics can lead and create the improvement.

Key Lessons

1.14 Experiment

1. APPLY was the consistent application of new processes and tools altogether within a baseline project. No innovative technology (from the point of view of the actual level in the organisation) had been introduced during this experiment.
2. Despite the quality and deepness of the preliminary definition of the new processes, the real scale experimentation is the only judge. The adoption is very slow and one should be prepared to iterate and to adapt.
3. In order to get a successful appropriation, it is highly recommended to involve the people who are clearly seen by everybody as the most knowledgeable in their field.
4. A major strength of the experiment and its dissemination was the balance between tools and procedures. New tools are simultaneously, an incentive to new practices, an immediate gain of productivity and a way to control process application.
5. Obviously, one of the most important factors of success is the commitment of the management. Controls and quantitative feedback of the application are the means to trigger the necessary corrective actions.

1.15 Business

1. Most of the project decisions from a business point of view are a function of three parameters: cost, time and quality. The first two are naturally quantitative and usually quite reliable. When no quantitative measure is available for the third one, the two other impose their law to the decision-

makers. Existence and credibility of predictive quality metrics are key to good decisions.

2. Requirements Management impacts are much more analysed before decision. Decisions on the technical content are taken with an accurate knowledge of the global impacts (cost, time and quality). As a matter of fact, time and cost budgets are much more likely to be respected for a given level of reliability.

Conclusions

The APPLY project was a very positive experience.

- Better mastery of the project.
  - **initial budget respected**
  - **delivery time managed within a 5% window**
- Better reliability of the product.
  - **number of remaining bugs divided by two for the same test effort**
- Positive cultural changes.

APPLY provides indirect benefits which are also very valuable. Compared to a project defined and run on an internal budget, the following benefits were experienced:

- A contract with the European Commission, as a customer, is very powerful to motivate the organisation.
- Thanks to the quantitative tracking process, the project itself is better defined and followed than a standard internal improvement project.
- The request for quantitative data is a very good incentive to improvements

Acknowledgements and thanks: Sincere thanks to the people who committed to the success of this experiment and who worked hard to get these achievements. The APPLY team directly involved in the experiment was composed of Jean Louis Camus, Bernard Coulange, Marielle Latapie, Marc Thirion and the authors. Thanks to all the people in the organisation who use the APPLY results in their daily work.

Creating a solid configuration- and test-management infrastructure to improve the team development of critical software systems

Author:Verstraeten Eddy
e-mail : eddyv@tessa.be
TeSSA Software NV
Clara Snellingsstraat 29
2100 Deurne
Belgium
URL : www.tessa.be

Abstract

This paper describes a process improvement experiment on the team development of critical information systems.

We have integrated PVCS from Intersolv for configuration management and SQA Teamtest for test-managment in our development environment. Our main development environments are Powerbuilder from Sybase and Visual C++ from Microsoft.

The formulated procedures have lead to an important improvement in quality and productivity.

This project could only be carried out with the financial support of the Commission of the European Communities, in the specific programme for research and technological development in the field of information technologies.

Executive Summary

Project

TeSSA Software NV is a developer of critical software systems for different markets (Shop-floor control, Warehouse-management, ERP-modules). The baseline project is a typical TeSSA Software NV product, situated in the market of paypoints.

The experiment, situated in the domain of configuration- and test-management, has contributed to the aim of being a software production house that delivers quality systems in time.

Project goals

By implementing the software control management the project manager can optimise the development process, in concrete terms:

- Cost reduction (10 - 15 %)
- Elimination of errors in an early phase of the process (1 in stead of 10)
- Quality improvement of delivered programmes.
- Reliability increase of installed programmes.
- Acceleration of the definite product delivery. (about 10%)
- Reaching these goals indirectly results in a better work-atmosphere for programmers, analysts, project managers and management.

Work done

- A quality manager was indicated permanently.
- An internal base-reference report is written, to situate problems and costs.
- The global IT company strategy was officially defined.
- Looking for other existing models we found SPIRE (ESSI Project 21419) promoting CMM and BootCheck, 2 very interesting projects, giving a wider frame for the global plan.
- Choice of appropriate tools :

- Version control system and configuration management : PVCS
- Testtool : SQA Teamtest
-Training in both products.

Results

At the end of the experiment, every employee agrees that quality and reliability of the software development process is improved significantly. First figures give a global improvement of 5%. This is less then expected (7 à 10%), but we believe that the positive influence in productivity and reliability will become more and more visible in the next years. The confidence in this experiment certainly helps to get a better working atmosphere.

The responses of the customers prove the confidence in the strategy of our company, working hard on the improvement of our internal processes and they see the first results of the new working methods.

Future actions

Now the procedures are consolidated and standardised to support the development cycle internally on the same LAN, the next step will be to extend the procedures to also support external employees.

With the help of our internal organisation with Lotus Notes Applications, the proceedings and the procedures are nowadays continuously internally disseminated.

Background Information

Company coordinates :
TeSSA Software NV
Clara Snellingsstraat 29
2100 Deurne
Belgium
URL : www.tessa.be
Contact person : Eddy Verstraeten
E-mail: eddyv@tessa.be

Strategy and products of TeSSA Software NV:

The information-management within companies is becoming more and more important to maintain a competitive position within their market sector. The policy of TeSSA is aiming to implement this information-management in a fast, yet affordable way, and still according to the customer's wishes. The core business is situated in industrial companies, where shopfloor control and warehouse management are crucial. TeSSA Software NV, as a supplier of software based IT systems in this vertical, but open, high competitive market, has worked out the global strategic plan in which the improvement of the software quality (and therefore the software production process) is one of the main issues.This PIE fits in this global plan and will help the company going to assessments and ISO9000 certification.

The project used in this PIE:
The baseline project is a typical TeSSA project, strategic for the future. It consists of a paypoint, that controls the renting of lockers. Up to 40 lockers are controlled by one control system. Up to 100 paypoints can be connected and installed in one railway-station (already installed pilot-systems : München, Hannover, Braunschweig).The control system, that fulfills all the functionality needs, is designed in an heterogeneous environment, that will be described in par 2.3.3.

Objectives

By doing this PIE our software engineers will apply a software control management. This is the key to **team productivity** on the LAN. (People are also developing at home.)

Without this control management, they experienced the following annoying incidents:

Fixes get lost, code is overwritten, multiple versions exist, duplicate or conflicting copies exist, failures in production. 20% of total time is spent to this kind of problems.

Implementing organisational procedures in the domain of configuration and test-management will help the project leader to control the development process.

Not only the thinking up and writing down of these procedures, but also the creation of driving forces behind the "quality-consciousness" of all people in the organisation, are important. Therefore internal sessions will be organised for all employees.

The implementing of these procedures will be carried out with the help of existing tools (ex. PVCS, PB 5.0 (which has version control options), SQA Teamtest, …).

This experiment will not only eliminate errors (1 in stead of 10), but also increase the productivity with at least 10%.

Our people, installing the product on the production site will be more confident and automatically a better work-atmosphere will be the result.

Starting scenario

Till mid-term there has been no assessment for software practices and we know we have a rather immature level of software development throughout the whole production process.

State of software development in general

This section should outline the initial circumstances at the beginning of the experiment.

1. Analysis

Customer interviews and reports, A functional description, Data flow diagrams (Yourdon - tool : now System Architect), Building Database (Entity Relationship Diagrams - tool : now System Architect)

Weakness: NOT ENOUGH INTEGRATION, NO TEST SCENARIO’S.

Prototyping

Powerbuilder: different painters TeSSA Software NV object library for Powerbuilder Third party libraries for other useful functionality.
Weakness: NO VERSION CONTROL.

Design

Technical description Building common libraries

Weakness: No VERSION CONTROL. NO TEST SCENARIO’S.

Programming

Microsoft C, C++, Powerbuilder with object oriented libraries. TeSSA Software NV common library Other useful third-party-libraries (e.g. Communications Greenleaf, ...)

Weakness: NO VERSION CONTROL, NO CONFIGURATION MANAGEMENT, NO AUTOMATED TEST-PROCEDURES.

Testing

Manual procedures Without any tool.

Weakness: NO AUTOMATED PROCEDURES.

Documentation

Manuals written with MSWord and a helptool to get on-line windows help.

Weakness: NO REAL VERSION CONTROL.

Business environment of the experiment

Looking at the business in which the experiment takes place now: The baseline project is a typical TeSSA Software NV project, strategic for the future. It consists of a paypoint that controls the renting of lockers. Up to 40 lockers are controlled by one control system. Up to 100 paypoints can be connected and installed in one railway-station. (Already installed pilot-systems: München, Hannover, and Braunschweig)

The control system, that fulfils all the functionality needs, is designed in a heterogeneous environment.

Technical environment of the experiment

The **running system** contains the following items:

The operating System is NT on a 486 motherboard. The underlying database is Sybase SQL-Anywhere, with the replication technology.
The network topology is Ethernet. The PLC from ED&A (Belgian company), that controls the locker doors, is connected through RS232.
The magnetic card reader is also connected through RS232.
The pay-systems (coin-box, …) are connected directly through IO

The **baseline production process** of this product:

Analysis with DFD-diagrams, database design with ERDiagrams, prototyping with PB.
The code for the PC-programs is written in C, C++ and Powerbuilder (GUI).
The code for the PLC is also written in C.
The code on the magnetic card reader is written in Assembler. Version control is carried out by copying sources to specific directories. No written procedures for synchronising the team work. No real test-plan, so the test-cycle depends on the programmer, who will test his own changes.

In **the experiment** the paypoint system is improved. We added some new functionalities:

Extension to other pay-systems (electronic monnai)
And we improved the existing functionalities:
Improved Object Oriënted implementation of a number of modules (previously written in C)
Improved integration of the PLC-network
Integration with latest versions of NT 4
Using multi-threading.

Organisational environment

TeSSA Software NV is only a small company doing a lot of tailor made projects.
People are the most important resource in this type of company. So the most difficult organisational part was “the spending time policy”. We are aware of the fact that this experiment only costs time in the near future, but will deliver quality and even will save time in the long run. Creating a new unit and indicating a full-time quality manager, being the driving force behind the whole project could only solve it.

Another important organisational issue in this kind of projects is communication and documentation. In our global company administration we’re using applications in Lotus Notes for many years. This document driven system is helping us a lot in communicating and not losing documentation. In this project all generated procedures, charts or any documentation will be saved in this Notes database.

Many years ago we started with version control in our DOS environment. With this history of version management, we had no problems introducing the PVCS tool in our new environments.

With the test-methods, there is more resistance. People are holding their traditional way of working. With the seminar, given by the test-evangelists of PS-testware, most of the people are seeing the real need for introducing the testing methods.
They were told testing does NOT mean demonstrating there are NO errors in the software, but to find as many errors as possible in a short time.

The people working with real object oriented environments are easier to convince then the classic procedural programmers. Automatically everyone starts to think in objects, because the testing of small isolated objects is a lot easier then testing a complex collection of functionalities.

Since most of the TeSSA Software NV-people are technical engineers, having a lot of experience in the software development, they can manage these new procedures. It’s another way of thinking, with a lot of standardised procedures in all phases of the development. They can produce software with higher quality.

The baseline project was mainly developed by 3 technical engineers. Two are not involved in the PIE, the 3rd engineer is now the projectleader of the second version of the project. He is not actively involved in the PIE itself. Of course being the project leader he’s the best judge of the improvements of the end-result.

Work Performed

Organisation

TeSSA Software NV is a small company of 15 people.
Two people are in charge of the strategic management of the company.
The quality manager did get a full time job, in the first year his time was spent entirely on this PIE.
Of course even after this PIE, the workgroup “Global Plan” remains a very important workgroup. This workgroup, mainly operated by the two mentioned persons in the diagram, steers our operational environment. Depending on the treated subjects other people of the company are involved in the discussions. Together with E. Verstraeten this workgroup must keep the global overview of our strategic direction.
Two workgroups were (and still are) active in the perspective of this PIE.
Workgroup Config:
Integrating PVCS (version and configuration management) in our different environments. The different environments are MS Visual C, C++, Visual Basic and Powerbuilder. The ultimate goal is to have similar procedures for each of these environments. Each environment has one active specialist in the workgroup.
Workgroup Testing:
Exactly the same goal as the first workgroup, for SQA Teamtest, our testing tool.
Under these workgroups is the developing personnel, utilising the procedures of this PIE. Their experience and remarks are treated in the different workgroups.
The job of quality manager was made permanent, and the workgroups continue to improve procedures.

Technical environment

Equipment is evolving quickly in the IT world. Tools require more and more powerful equipment. To make this PIE successful, everyone working on this project, with the new tools, has an equipment with minimal a Pentium 200 processor and 64 MB RAM.

Two new tools were introduced:
PVCS: Version control and configuration management.
SQA Teamtest: Test system.
Communication and reporting is established using an application written under Lotus Notes. All procedures are written down in this application and everyone in the company can access this database with full-text search engines to easily find the required procedures.

Training

PVCS:
The first training was given at the site of Intersolv. Two of our people joined the course and got a detailed technical explication.
SQA_Teamtest:
A specialist of PS-testware gave the general course in testing philosophy and testing ethods at the site of TeSSA Software NV for all the developers. Our quality manager himself had a course about the technical details of SQA Teamtest.
In February PS-testware consulted during 4 days on test procedures.

Results and Analysis

The most important motivation for this PIE was stated in our project programme as follows:

“The new development technology and methodology must improve the maintainability and the reliability of the produced software modules.”

Upon completion of this PIE everyone (management, workgroup members and programmers) agrees that we did a very good job, but the work is not finished yet.

We will make a comparison between the results of the base-line project and the later versions of the software. This baseline-project consists of a number of modules. These modules evolve with the requirements of the customer. Further evolutions were compared with the baseline (costs, errors and reliability)

Technical

A very important outcome of this PIE was the global strategic plan. In this plan we started from the current status of software development. Main objective of the global strategic plan was the definition of a new framework of a life cycle model for the software development process of our company. In this global framework we had to integrate the configuration and the test management.

Configuration management

The new version control procedures are developed and implemented in the new framework of our production process. This was very important because less than 30% of a developer's time is spent in programming **new** software. Most of the time is occupied with existing code. Certainly in team development, time and money could easily be wasted on redundant efforts, accidentally overwritten code, etc.

Introducing PVCS and the procedures takes on the time consuming and non-creative tasks of project management and version tracking, leaving developers free to concentrate on code. Ultimately, it also leads to better quality software, so developers spend fewer resources on code maintenance duties.

Test management

The test management is "completed" for the MS Visual C++ environment, but is still going on for the Powerbuilder environment.

The methodology used for "Structured Software Testing" is the V-model. The implementation of this model consists of setting up a Test-assignment, writing the testplan with the development of a test requirements hierarchy, making the test designs and testscripts and at the end execute the tests.

This whole process is managed by our new tool SQA teamtest.

Using the V-model it is really remarkable how many errors are detected in the early phase of software modification. Analysing a project and simultaneously implementing the test scenarios for this project let us detect some logical errors in a very early phase of the project.

Results

Project A: In the baseline project we had a specific software module that implemented a device for cash payment. In the history of our time-tables this project was extracted and evaluated in working hours and reported errors before and after installation of this module.

Project B: In the same project, we developed a new software-module: implementing other payment methods with creditcards. Working hours and error-reporting was registered consciously.

The functionality, the degree of difficulty, the impact on the main system and the global implementation of both modules may be compared.

The engineers developing project B were not the same engineers developing project A. This is an important aspect for the analysis (we can state that people starting project B had the same level of experience as the people ealier working on project A).

The modules have no impact on each other.

An end-user of the system uses one of the payment methods. Both methods can't be used at the same time within the same transaction.

The results of our analysis are given in the next table :

	Baseline-project A Implementing Cash-payment device	%	After PIE-project B Introducing other payment methods (credit-cards)	%
Estimated Project Man-hours	300		400	
Real Man-hours	334	111,33	411	102,75
Hours spent for analyzing	54	16,17	87	21,17
Hours spent for programming	182	54,49	201	48,91
Total Hours spent for testing	67	20,06	100	24,33
Installation time	31	9,28	23	5,60
Number of errors in early phase	14	4,19	21	5,11
Number of errors after installation	18	5,39	3	0,73
Total errors	32	9,58	24	5,84
End-user impression	6		8	
Cooperators	5		7	

The estimated workhours for both projects were respectively 300 and 400 hours. Project B was only 2.75% out of budget, while A was more then 11% out of budget.

But much more important are the 2 last results. The quotation of customers evolved from 6 to 8. We had a questionnaire for the customers, concerning global satisfaction. The average of the quotation with project A was 6, with B 8. Especially the question concerning the satisfaction about the installation on site evolved dramatically from 5 with project B to a 9 with project B.

The reliability of the software is also expressed by the quotation of the employees, evolved from a 5 in project A to a 7 in project B.

In the table we also find a shift from programming to analysis and testing. This can be logically explained by the V-model in testing. More time is spent in the analyse-phase on developing test scenarios. These scenarios also lead to the discovery of errors in an earlier phase. Besides the total numbers of errors was reduced from almost 10% (errors divided by total working hours) to less than 6%. The errors reported in both columns correspond only to the specific project. The 6% errors in the experiment (project B) correspond only to the new analysis, programming and implementation.

The reduction of errors after installation from more then 5% to less than 1% is remarkable and this explains the reduction of installation hours at the customer site and also stands for a great customer satisfaction.

From a cost perspective, we can say that first of all the estimation of the project was more correct, due to a better understanding and standardisation of our software life cycle. The effect of this improvement is difficult to quantify.

With the configuration and test management in project B, a second improvement was realised, so the global cost improvement is at least 5%.

Business

The cost for travelling (to implement our software on site) decreased clearly because the implementations can be done quicker and easier.

Another unexpected result of the version management is the better information towards the end user. The project manager knows exactly what we have planned to improve in next version(s) of the software project and everyone knows the exact definition and the influence of this version in the different modules. The document with all this information is also sent to the customer. He knows what to expect from the new version and when to receive it. This means that the product quality has increased dramatically and time-to-market is much quicker. These are two top issues in a very competitive market.

The traceability, resulting from this version control is very useful for our management, to measure the effectiveness of our programmers and analysts. For management this feature becomes a real important tool. Also the project schedules are more reliable.

With the test procedures implemented from the 1st phase of the project, many errors are detected very early and the result will be a better customer satisfaction, as shown in 4.1 We did an oral inquiry at our most important customer sites, equipped with software modules that were programmed with the new procedures. The questions were asked both the project leader and the customer’s end user.

We also expect a reduction of the support calls, because the testing procedures eliminated a lot more errors. This also ensures happier customers.

Organisation

The creation of a new unit with a quality manager with the mission to coordinate all improvement projects is a real enhancement to our organisation. This gives the company the driving force for a general quality improvement. The use of this framework clarifies the roles in the development process

A better customer satisfaction automatically results in a better atmosphere. Our people have less stress and feel the good influence of structured working procedures.

Culture

Programmers must be convinced that testing means you want to reveal the errors that are in the software, it is the process of executing a program with the intent of finding errors. Testing does not mean demonstrating there are no errors.

The baseline project is a software application that has to be installed very often. Everyone agrees that with this kind of applications version control and testing methods are a need and will gain money in the long run.

But our company does many projects as tailor-made projects. I hope that with this experience everyone will be convinced to take over the same methods in the environment for tailor-made projects. We hope the procedures and the system will be so simple and efficient that also in the tailor-made projects the testing philosophy will win.

The driving forces, the management and the quality manager, try to introduce the expectation of improvement on all levels. The creation of the improvement culture for all the levels and all the different projects gives new ideas on all levels. All these ideas can be discussed and eventually integrated in the working procedures.
To improve the acceptance level of the procedures, they are being adapted to be practical (otherwise they add too much bureaucracy.)

Skills

Version control is a pure internal technical issue. It doesn’t require special other skills then our people have today. It’s just getting to know the procedures. Once the people working on the projects know the procedures, they can easily integrate these procedures in their current way of working.

The testing workgroup clearly showed that the process of determining the requirements of the customer, the technical analytical process and the resulting technical documentation are very important. These issues demand other skills and knowledge then the pure knowledge of information technology. Some of our people have the opportunity to climb up to a higher project task level.

Another result is the teamworking-spirit – The coaching sessions and the participation in the workgroups gives motivated people.

Conclusions

We achieved the technical objectives. A global improvement of 5% on the studied projects is a success.

The experiences were very positive and boosted the team spirit as well as the global strength of the company.

From the business point of view we see that the effect on the client perception is the one expected. He is more confident in our products than before this PIE. Also the employee perception is the one expected, namely a greater confidence in the installed products.

The final balance of the results per invested cost are really favourable thanks to the good acceptance inside our company and at the customer site.

Future actions.

At the end of this PIE the configuration management is implemented well for teams, working on the same server. Since we want to go to modern methods of teleworking and working with people on distance, we must extend the configuration procedures to that kind of coöperation. In the near future the procedures must work over the Internet. This will be a new experience.

The test management is completed for our Visual C, C++ environment but is still ongoing in our Powerbuilder environment. Automating these test-procedures was really underestimated and will be going on for a few months. The result of test procedures in the analysis phase (following the V-model) however are remarkable. We will continue the implementation of these test methods and certainly improve the automation process of testing the programs itself (based on the scenarios as defined in the analysis.

Due to the encouraging results achieved so far we are already looking for other improvement projects in the wider framework of the CMM model. First we have to continue the implementation of the testing process. After stabilising these new processes we intend to perform a formal assessment and update the improvement process plan conform to CMM.

One of the goals of the company remains to have an ISO certification.

References

SPIRE : SPIRE consortium (ESSI Project 21419) promoting CMM. The SPIRE partners :
Objectif Technologie – Arcueil Cedex, France
CTA – Hatfield, United Kingdom
FIRST Informatics – Patras, Greece
Intrasoft – Athens, Greece

BootCheck : "Assessing your software process" by the
European Software Institute – Parque Tecnologico, Zamudio, Spain
Bootstrap Institute

An industrial experience in improving the software process through domain analysis

Andrea Valerio
The Software Production Engineering Laboratory, DIST,
University of Genova, Genova, Italy - E-mail: Andrea.Valerio@dist.unige.it

Massimo Fenaroli
Thera S.p.A., Brescia, Italy - E-mail: mfenarol@thera.it

Luigi Benedicenti
Faculty of Engineering, University of Regina, Regina,
Sascatchowa, Canada - E-mail: Luigi.Benedicenti@uregina.ca

Giancarlo Succi
Department of Electrical and Computer Engineering, University of Calgary,
Calgary, Alberta, Canada - E-mail: Giancarlo.Succi@enel.ucalgary.ca

Abstract

Nowadays software applications are present everywhere and producing them is a hard task: software firms are facing an increasing demand for new applications with higher quality and lower prices, but it is often difficult to exploit this business chance. This paper presents a case study regarding the adoption of domain analysis inside a software firm with the purpose to improve the software process and to introduce software reuse practices. Domain analysis is the process of identifying, collecting, and organising all the artefacts used in software development in a particular domain with the goal to make them reusable. This methodology proposes to enhance the software process of a firm, augmenting productivity, reducing time to market and improving the quality of the delivered products. The case study took place in a real industrial project, integrating the software development process of the organisation with the new practices coming from the introduction of domain analysis. We discuss the results we collected through the measuring program that we set up to monitor the experiment

execution. The evaluation we made and the quantitative figures we obtained shows that domain analysis, and reuse practices that it fosters, can take real benefits to the organisation. In the last section of this paper, we investigate the implications that the experiment had in the organisation and we describe future actions.

Introduction

The actual situation of the information technology market is characterised by an ever-growing demand of new software applications with higher quality and lower prices. Software firms are not always ready to exploit this favourable situation: they often do not have the capability to deliver the software applications requested by customers with the quality level, the price and the time-to-market expected by users. This situation, that some identify with words such as ‘software crisis’, creates a stimulating environment where new technologies and techniques are continuously invented and proposed. The need for robust and controlled development practices that allow software firms to satisfy customer requests and to survive in the IT market, has grown for years. Today most firms are considering the adoption of software process models and software engineering practices in order to formalise their production processes. Although difficult and slow, this trend is strongly committed to by software practitioners and managers on the way of some experiences done by first movers.

Domain analysis processes existing software application in a given domain to extract and package reusable assets. Systematic software reuse, a technique that promise relevant benefits, requires an understanding of the work done in past projects: in particular a major problem concerns the creation of assets that can be reused in a context different from that where they have been developed. In this view, domain analysis is a fundamental activity integrated in a software process based on reuse. This, in turn, reflects into business process improvement, and strong commitment to it is a direct consequence of the business need of the organisation.

This paper describes an experiment concerning the introduction of domain analysis inside an Italian software company, Thera S.p.A.. The goal of the experiment is to improve the software process, achieving an increased productivity and a higher quality of the products. At the same time, the introduction of domain analysis should foster and support the institutionalisation of a consistent set of reuse practices. This work aims to outline the implications that the introduction of a specific software engineering technique, i.e. domain analysis, can have, in a similar way as a technology innovation action, on the software development process, basing on the experience we made in a industrial project. We present a detailed analysis of the quantitative data collected during the experiment and we discuss the impact that this experiment had on the organisation and on the people. The results we achieved and the lessons we learnt demonstrate the benefits that domain analysis may have in the software development process, in particular in relation with the increase of reuse efficiency.

Domain Analysis: the State of the Art

Domain analysis is an activity occurring prior to system analysis. It aims to identify features common to a domain of applications, selecting and abstracting the objects and operations that characterise those features. The first definition of domain analysis was introduced by Neighbors as "the activity of identifying the objects and operations of a class of similar systems in a particular problem domain" [Neighbors81]. In general, domain analysis should support extraction, organisation, analysis and abstraction, understanding, representation, and modelling of reusable information and assets from the software process [Prieto90], with the purpose of making them available (reusable) for the development of other products in the same or in different domains.

The underlying idea is that different products inside the same application domain share similarities. Such similarities generally imply that the products in the same domain could use similar components. In the words of [Arango93], a domain can be viewed as a collection of similar applications. Such applications do not need all to exist yet. The objects of study are applications. The result of domain analysis is a taxonomy of applications based on differences and commonalties. House builders take a similar approach when they build a new house: they identify the main structures of the building among few predefined ones. Then they choose among different variants and they build the house using the structures and the component identified.

In these last years, many authors have proposed different domain analysis methodologies. Each domain analysis method proposed in literature has its own peculiarities due to the specific problem to be solved and to the approach adopted to solve the problem, that can be for example problem-driven or application-driven, reuse-oriented or knowledge-representation oriented. Arango proposes a comprehensive survey on domain analysis [Arango93], and Wartik and Prieto-Diaz describe an interesting comparison of several reuse-oriented domain analysis approaches considering the context in which they were conceived [Wartik91]. Basing on the activities that are shared by the different domain analysis methods, it is possible to identify a general model for the domain analysis process. Along the lines of [Arango93], the general model can be structured into four main phases, each phase constituted by different activities:

- *Domain Characterisation and project planning*: the first step of every domain analysis method is a preparation activity. It aims to collect the minimum information concerning the problem that allows to decide if it is worth to deal with it and try to solve it, or if it is not feasible to go on with the process.

- *Data Analysis*: the necessary information for the analysis is collected and organised, then the analysis exploits domain commonalties and variations.

- *Domain Modelling*: the purpose of the modelling phase is to complete the previous analysis step, building suitable models of the domain. It deals with modelling common aspects in the domain, refining domain models encapsulating variations possibilities, defining frameworks and general architecture for the domain, describing the rationale beneath domain models and tracing technical issue and relative decisions made in the analysis and modelling process.

This phase can be considered the core activity aiming to produce reusable assets, such as components, frameworks and architectures. The difference between domain modelling and system modelling lies in the target chosen: in system modelling, it is the specific software system that has to be built; in domain analysis, it is a class of similar systems in a specific application domain.

- *Evaluation*: its purpose is to verify the results of each step of the domain analysis process, identifying possible errors done in building the model, and to validate the results against requirements and user expectations.

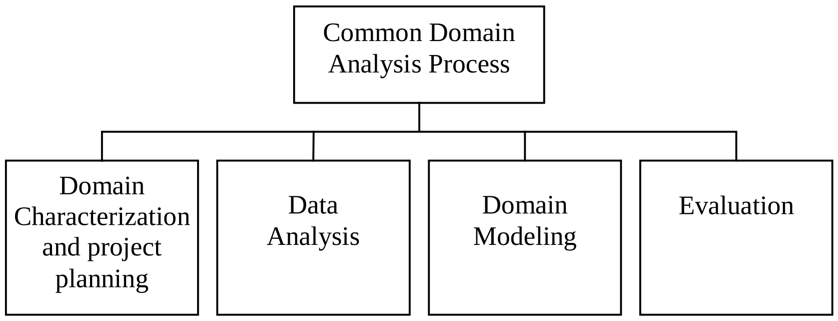

Common Domain Analysis Process
Domain Characterization and project planning
Data Analysis
Domain Modeling
Evaluation

Figure 1: The general model for the domain analysis process.

Integrating Domain Analysis in the Software Development Process

Domain analysis can be defined as: “a process by which information used in developing software systems is identified, captured and organised with the purpose of making it reusable when creating new systems” [Prieto90]. During software development, different information is produced, and the software product delivered is only a part of this heap of data. One of the main goals of domain analysis is to analyse all this information aiming to exploit and reuse most of them in present and future software development projects. Domain analysis fosters process improvement through software reuse: it supports the identification and definition of information and components that can be reused in applications and in contexts different from the ones for which they were originally conceived.

Walking side by side with software reuse, the emphasis in domain analysis has moved from code analysis to the analysis of every kind of information produced in the software process, with the goal to identify and define high level reusable artefacts, such as frameworks and architectures. In this perspective, the domain analysis process became a fundamental part of a global software engineering process whose purpose is to produce new applications reusing components, frameworks and information from past projects and aggregating them following the model proposed by a general domain architecture. This leads to an improvement of the business process as a direct consequence of the improvement introduced in the development process. Software reuse and domain analysis foster standardisation and mass production by simplifying the assembling and configuration activity. Domain analysis helps in analysing and tracking past products and projects, contributing in the definition and maintenance of repeatable process practices. Domain analysis supports the identification of variants and possibilities for later customisation and specialisation, encapsulating these aspects inside domain models (such as frameworks and architecture) which can be customised into new software products that meet user needs. One more benefit of domain analysis is connected to the possibility of classifying and structuring knowledge and expertise acquired in past projects.

The SELDOM process improvement experiment

SELDOM is a process improvement experiment whose main objective is to improve the software production process of an Italian software organisation, Thera S.p.A., with the introduction of domain analysis and design frameworks. The introduction of a formal domain analysis method and sound reusable component design principles (design frameworks) aims to reduce the effort of producing new products within the same domain, by analysing and predicting commonality and variability within past, current and future applications. The SELDOM project focus on the improvement of the business process. It is a direct response to the following needs: improvement of the quality of the development process (which in turn leads to the improvement of the quality of products), improvement of the structure and management of the production process, increase of productivity and reduction of the time to market for new applications.

The context of the experiment

Thera’s development process is based on the extension of the waterfall model to support refinement and incremental production, formalised around the object-oriented technology. The process of capturing expectations and requirements is managed "ad-hoc"; in the analysis and design phase the Booch methodology is employed, supported by the Rational ROSE tool. The design is based on the IBM San Francisco framework and the implementation is done with the Java object-oriented language. The development environment is JBuilder (Borland).

The domain analysis methodology introduced during the project is a customisation of the PROTEUS [Proteus94] and FODA [Foda90] approaches. These general methodologies have sound, detailed, and easy available documentation. The domain analysis method proposed by PROTEUS is the best match for our context: an incremental software process based on object-oriented techniques. It represents commonalties and variants with the concepts (common to most of the object-oriented techniques) of constraints and associations, generalisation and specialisation, abstraction and aggregation, multiplicity and metadata/metamodel. FODA is interesting because it focuses on user-visible aspects of software systems, the ‘features’, and highlights some useful concepts and deliverables. We adopted in the SELDOM project a customised version of PROTEUS, adding to it the peculiar domain-characterisation focus of FODA (specifically, the gathering of domain requirements and user expectations) and the documentation strategy it expresses. Both PROTEUS and FODA aim to produce domain architectures: we expanded this activity introducing the related concepts of frameworks and general domain architecture, delivered as reusable assets in the domain.

Description of the experiment

The experiment we performed consisted in the introduction of domain analysis in the software development process concerning the specific baseline project described in the previous section. The baseline project adopted for the experimentation of domain analysis was the development of a software system named "Logistics Management". It concerned the development of a proprietary product, whose main goal was to help a manufacturing organisation to plan and control stocks, sales and purchases. The language chosen for the implementation was Java. Besides, the San Francisco framework sold by IBM was the underlying software architecture over which the product has been built.

During the first phase of the experiment, ‘Training, study and methodology adaptation’, the personnel directly involved in this PIE was trained on the basic aspects of domain analysis and framework design, aiming to provide them the necessary knowledge to perform the experiment. Then, an in-depth study was conducted to understand how to adapt domain analysis methods (PROTEUS and FODA) to the software development environment of Thera.

The second phase was the ‘Application of domain analysis to the baseline project’. We used the domain analysis method adopted for the experiment to analyse the logistics management domain. The goals were to identify the common features of the applications present in the domain, to formalise the user requirements, defining the object-oriented models of the entities in the domain and to identify common design frameworks. The results of these activities were then applied in the baseline project.

The three main tasks that compose the application of domain analysis were ‘Domain characterisation’, ‘Domain model definition’ and ‘Design frameworks development’.

The first activity, ‘Domain characterisation’, dealed with the identification and the classification of the information concerning the context of the domain under consideration.

In the second activity, ‘Domain model definition’, the requirements and the information collected in the previous step were analysed and object-oriented models were built, following the prescriptions of the Booch methodology. The Object Model was designed to represent the object structure of the domain. It was refined applying the concepts of generalisation and aggregation, identifying commonalties and hierarchies among classes, resulting in a comprehensive model that represents the common aspects of the system. Aggregation is one of the basic mechanisms in the production of software applications through asset composition.

A Dynamic Model supported the Object Model, providing a dynamic description of the behaviour of the system in terms of interaction between objects, and representing the evolution of the system in response to external events. The description of the variant aspects of the system completed the modelling activity.

The third activity, ‘Design frameworks development’, took as input the object-oriented models produced in the previous step and refined them with the purpose of creating reusable components that feed the software projects. Classes were grouped into separate categories, basing on their relationships and interconnections. Classes having strong cohesion among them and presenting a scarce coupling with classes from other groups constitute a good candidate for framework. Considering the variable aspects identified during the previous step, domain models were modified in order to incorporate the required variation points, following the indications of suitable design patterns such as: «Factory», «Template Method», «Composite», «Decorator», «Bridge», «Observer» [Gamma95].

Concurrently to the experimentation of domain analysis in the baseline project, we performed the ‘Data collection’ activity, which was part of the ‘Data collection, analysis, comparisons and dissemination’’ phase and consisted in the collection of experimental data during both the experiment and the baseline project. This monitoring program aimed to collect statistics on development effort, size of code and reuse of modules as numerical indicators of the performance of the process. The purpose is to assess whether measurable goals planned at the beginning of the experiment have been achieved. The final comparison of the situation before and after the introduction of domain analysis is yet not complete.

Analysis of the results of the experiment

During the whole experiment, we carried out a measurement program collecting two kinds of information: the effort spent during each phase of the development process (in person-days) and product metrics (number of modules developed from scratch, number of module reused, etc.). We relied on a tool that we developed in-house for the collection of effort data related to daily activities performed by the personnel. Each person, at the end of the day, introduced his personal data in the tool and linked the data to the logical subsystems (presented with different levels of aggregation) of the baseline project under development. In our view, a module is a logical aggregation of one or more basic business classes with the service classes strictly related to them added by San Francisco, such as as the ‘factory’ class, the ‘controller’ class, the ‘stub’ and ‘skel’ classes [Gamma95]. This is motivated by the use of the San Francisco framework, which provides a semi-automatic generation of code associated with support objects for each logical class realised. We roughly estimated that about 30% of the code lines were produced automatically by San Francisco (we do not have a parser suitable for such detailed a measure).

The data we collected can be summarised by the following indicators:

580 logical modules were developed during the project (A), corresponding to 5.563 physical classes for 555.690 lines of code (LOCs), free of 445.587 line of comments.

205 modules were developed for-reuse (B), consisting in about 238950 LOCs. These were produced as reusable components to be assembled in the production of other modules; these modules built the reusable library for the Logistic management system.

13 modules were produced as ‘abstract’ superclasses (C), and 13 modules (accounted in the next figure) as ‘first-implementation’ from the superclasses (produced by composition).

130 modules were developed by composing different reusable library components (D), reusing the knowledge and following the implementation guidelines exploited with the development of the 13 ‘first-implementation’ modules.

232 modules were developed ’ad hoc’ and they have not been reused (E).

369 times reusable modules were reused in the project for the composition of the 130 modules cited above. We calculated this value by a manually estimating in the design documents an average of 1.8 reuses for each reusable module present in the library.

Considering these numbers, we calculated the following measures:

- Number of reusable modules developed / total number of modules developed [B / A]: currently in Thera this ratio approaches 0.15, because until now there has been no incentive to develop reusable modules. Before the experiment, we planned to shift this ratio near to 1 after the complete instalment of the domain analysis process and the institutionalisation of reuse practices, while we expected to reach a rate of 0.35 at the end of this first experiment.

- LOCs in reusable modules developed / LOCs in all modules developed in the project: the achieved rate is 0.43. It is better than the rate regarding the number of modules, because the code within the reusable modules, embodying generalised behaviour, is more consistent than the code needed to adapt them to the specific context.

- Number of modules developed with reuse / number of modules developed from scratch [D / (B+C+E)]: currently this ratio is between 0.1 and 0.3. We expected to achieve a ratio between 0.3 and 0.5 at the end of the experiment: the result we reached, a ratio of 0.29, is slightly less than we expected. A possible explanation is that reuse has a long term benefit, i.e. the reusable components have to be reused several times before a real cost decrease is achieved.

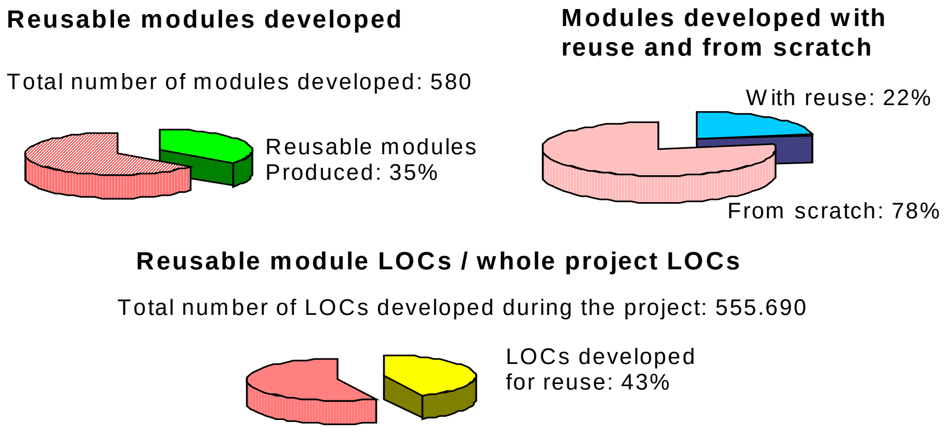


Figure 2: Pie charts for the reuse measures collected during the experiment

Qualitative analysis of the experiment and its impact on the organisation

From a software engineering point of view, the main qualitative results achieved until now can be synthesised in the following points:

- the greater level of formalisation of user requirements that has been necessary in the first steps of the domain analysis activity in order to characterise the domain and define its boundaries has allowed personnel to gain a deeper knowledge of the application domain, with positive effects in the modelling and framework design activities;

- the introduction of a comprehensive framework, such as IBM San Francisco, that provides a basic set of object-oriented infrastructures and appreciative components that can be extended, has allowed to concentrate the resources on the problem domain (rather than on technological services) and to base the software production process more on the reuse of components rather than on the development from scratch of new ones;

- the definition of the domain models has contributed to shift the main effort from implementation towards planning, leading to a better modularization of the system, a robust architecture, a sharper documentation, resulting in a higher quality of the analysis and design assets;

- such higher quality, together with the organisation of a development process more focused on reusing components and specialising frameworks, has determined an higher quality of both the software process and the developed products.

Considering the organisational aspects, the experiment had a significant impact on the methodological point of view, but, on the other hand, it had a smaller influence on the organisational structure. This could be due to good practices already deeply rooted inside Thera for correct and efficient organisation of the development process.

The main impact the experiment seemed to have was a strong spur towards a greater integration of the project team members and a tighter collaboration between domain experts, analysts, designers and software engineers. Now their specific activities are much more joined by a common background: the domain models and the design frameworks directly derived from them.

The technical knowledge obtained by the training program and the practical experience personnel is gaining from the execution of the experiment is leading to an increase of the skills of the staff involved concerning the domain analysis methods and framework design techniques.

This has a positive impact both on Thera and on the personnel itself. People are strongly motivated because of the benefits that the methodology brings to the development process in general and to their work in particular. Besides, they are conscious of the relevance of their role in the evolution of Thera’s development process and they are satisfied for their increased skill and expertise in up-to-date technologies. This turns into a competitive advantage for a software firm like Thera, which is based on the work performed by its personnel.

Conclusions and future work

In this paper we outlined how the introduction of domain analysis in the software process can influence and improve the whole organisation and the software development process in particular. Integrating domain analysis in the software process requires a first assessment and description of the development process in order to identify the current procedures and practices. The model produced for describing the organisational structure represents the starting point for the integration of domain analysis inside the development environment. We present a process improvement experiment regarding the adoption of domain analysis inside an Italian software firm with the goal to improve the software process and to introduce software reuse practices. The SELDOM project is an experiment guided and motivated by the business need of the firms: through the adoption of a specific technology, domain analysis, we want to evaluate how software engineering can improve the capability and maturity of the organisation.

The results we achieved, even if not definitive due to the still ongoing analysis, seem to confirm that the introduction of domain analysis in the software development process can really improve it and foster reuse practices at the same time. The quantitative data we analysed are not complete, but they give precious hints on the benefits we reached. This data has to be considered with special care because it refers to a single subsystem where domain analysis, frameworks adoption and reuse practices were optimally applied. However, the average reuse rate achieved in the experiment is still a very positive indicator of the benefits that domain analysis, combined with reuse, can introduce in the organisation.

The average results we obtained confirm the expectation we had before the start of the experiment. Moreover, it has to be considered that domain analysis and reuse have a return on the investments that is achieved only in the medium and long term. This is mostly because producing components for reuse and building a domain reusable library is a costly activity in the short term, and the benefits can be appreciated only when the reusable components are effectively reused from 2 to 5 times [Favaro96]. In this perspective, the evaluation of the results we achieved with SELDOM can not be considered finished, but we will integrate and carry on it with the analysis of the data coming from the current and future projects.

In the future, we want to extend these experiments concerning the introduction of software engineering in the development process of Thera. In particular, we intend to build on the core competencies we gained in this experiment with the goal to shift our development process towards a compositional model based on reuse and domain engineering.

Acknowledgments

The SELDOM project is co-financed by the European Commission under the framework of the Esprit ESSI PIE (reference: 23838).

Bibliography

[Arango89] Guillermo Arango, *Domain Analysis - From Art to Engineering Discipline*, in Proceedings of the Fifth International Workshop on Software Specification and Design, Pittsburg, PA, 1989, pp. 152-159.

[Arango93] Guillermo Arango, *Domain Analysis Met*hods, in Software Reusability, ed. W. Schaeffer, R. Prieto-Diaz and M. Matsumoto, pp. 17-49, Ellis Horwood, New York, 1993.

[Basili94] V.R. Basili, L.C. Briand, W.M. Thomas, *Domain Analysis for the Reuse of Software Development Experiences*, in Proceedings of the 19th Annual Software Engineering Workshop, December 1994.

[Favaro96] J. Favaro, *A Comparison of Approaches to Reuse Investment Analysis*, Fourth International Conference on Software Reuse, April 23-26, Orlando, Florida, 1996.

[Foda90] J. Hess, S. Cohen, K. Kang, S. Peterson, W. Novak, *Feature-Oriented Domain Analysis (FODA) Feasibility Study*, Technical Report CU/SEI-90-TR-21, Software Engineering Institute, November 1990.

[Gamma95] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Publishing Company, 1995.

[Neighbor81] Neighbors, J., *Software Construction Using Components*, Ph. D. Thesis, Department of Information and Computer Science, University of California, Irvine, 1981.

[Prieto87] Ruben Prieto-Diaz, *Domain Analysis for Reusability*, in Proceedings of COMPSAC 87: The Eleventh Annual International Computer Software and Applications Conference, IEEE Computer Society, Washington DC, October 1987.

[Prieto90] Ruben Prieto-Diaz, *Domain Analysis: an Introduction*, Software Engineering Notes, ACM, Vol. 15, no. 2, April 1990, pp. 47-54.

[Proteus94] Heweltt Packard, Matra Marconi Space, CAP Gemini Innovation, *Domain Analysis Method,* Deliveable D3.2B, PROTEUS ESPRIT project 6086, 1994.

[Wartik91] S. Wartik and R. Prieto-Diaz, *Criteria for Comparing Reuse-Oriented Domain Analysis Approaches*, In International Journal of Software Engineering

Using CASE to enhance service performance in Local Government: the CAPELLA project

EXPERIENCE REPORT

Karlheinz Kautz[1], Peter Kawalek[2], Matthew Keenan[3], Tom McMaster[4], Clive Walker[3], David Wastell[2], Michael Willetts[3], Chris Williams[3]: [1]*Norwegian Computing Centre, Oslo, Norway;* [2]*University of Manchester,UK;* [3]*Salford City Council and* [4]*University of Salford, UK.*

This paper reports the motivation, history and the interim findings of an ESSI Process Improvement Experiment (PIE) involving the deployment of a CASE tool in the IT Services Department (ITSD) of Salford City Council.

BUSINESS CONTEXT

Salford City Council, like each of the other 513 Local Authorities in England and Wales, is under constant and increasing pressure to improve the services it provides to its publics. Striving to maximise 'value for money' entails the constant re-engineering of service delivery methods and business processes. Nor is this any less the case today with the 'new' UK Labour Government than it was with almost two decades of its Conservative predecessors; the direct threat of Compulsory Competitive Tendering (CCT) may have receded, but the present Government's 'White Paper for Local Authorities' has in essence replaced it. This sets out a *Best Value* regime, requiring Local Authorities to put in place comprehensive and rigorous service plans that ensure conformance to the spirit as well as the letter of the regime. Once again Information Technology (IT) is seen as the key to the attainment of such transformations, and consequently there are increasing expectations of IT Services staff – that they develop and deliver solutions to ever higher levels of quality and economy. The CAPELLA experiment then is fundamental not only to better equip the service in delivering improved products, but also to assist in strengthening its capacity for coping with the changes that arise from new demands and expectations.

The IT Services Department (ITSD) has a mixed reputation within the local authority in the performance of its functions. In general terms, through analysis of customer satisfaction surveys, *Software Quality* is perceived as acceptable although *Timeliness and Budgeting Targets* were regularly exceeded. Interestingly, those software products where very strict deadlines are imposed through legislative timetables (for example the Council Tax System introduced in 1993) are invariably met. This lack of consistency was both troubling and heartening; *where the Service must succeed – it can, and does.*

The Development and Customer Services Business group, a sub-unit of the ITSD, is the major organisational unit involved in the CAPELLA experiment. The group's role is to provide a full range of development services, including *specification, analysis, design, development, implementation* and *support services* to internal departmental customers within the authority. In the early 1990's the unit had adopted the *Oracle* tool set as the chosen development product following evaluation of a number of options from various suppliers. It therefore seemed like a logical step to select the Oracle corporation's CASE tool (*Design/Developer 2000*) for deployment to ensure effective integration into the existing setting with minimum disruption to the present environment. To support the implementation and evaluation of the CASE tool, funding from ESSI was applied for in order to conduct a Process Improvement Experiment. This funding was obtained and the CAPELLA project was formally inaugurated in March 1997.

CAPELLA aims, objectives and organisation

The project's twin aims are to evaluate the business rationale for CASE tools use within the software development functions of local government, and to establish a culture of 'continuous improvement' in the software development process. Specific objectives set out in the approved ESPRIT Project Programme were:

- To achieve direct software process improvements in terms of development lead-times, software quality, and developer productivity.

- To reduce the cost of ownership of IT systems through higher engineering quality and reduced rigidity of applications, and the ability to better deal with future modifications.

- To use the features of CASE tools and methods to enable a 'total team approach' to software development, characterised by higher levels of customer involvement.

- To devise and validate an implementation strategy for CASE tools and methods, involving an in depth study of the human and organisational factors associated with the notion of 'technology transfer', in order to enable such change processes to be effectively managed and supported.

- To establish a culture of continuous improvement based on process-orientated, enterprise-wide views of software development.

The project was originally divided into three main phases.

The first phase consisted of creating the necessary capability for the project. This included the purchase and installation of the CASE tool, plus initial staff training in the use of the product. This phase also consisted of the establishment of the team structure to undertake the project with the creation of a '*Centre of Excellence'*; that is, those who would be responsible for technical advice and support on the use of the tool, to others in the development unit. Then, a baseline project was identified which was intended to develop the initial experience and capability in the use and application of the tool.

The second phase involved the creation of the necessary investigation and analysis frameworks on which to base subsequent evaluations. This included the production of a development methodology to underpin the use of the tool, and the development of structured frameworks for capturing hard and soft metrics on a range of development projects.

The third and final phase involves the actual experiment itself, including both internal evaluation and customer evaluation of the issues and impacts of the experiment.

It is important to mention that the project has been a collaborative venture between ITSD, the Universities of Salford and Manchester, and the Norwegian Computing Centre. The role of the consultants / researchers from these organisations has been to provide advice on CASE implementation and software metrics, supervise data collection and analysis, provide specific advice on experimental design, help write reports, and to provide general advice on organisational structures and the management of change.

INTERIM TECHNICAL FINDINGS

In this section and the next, the interim findings of the project (i.e. at the end of stage two) will be reported. Five technical themes of work have been carried within the project:

- development of a structured methodology
- development of an evaluation framework
- analysis of costs and benefits of CASE
- development of training materials
- development of a CASE implementation strategy

Methodology

In order to establish a firm base for the experiment, a series of in-depth interviews were carried out in the first two months of the project. These interviews were with users and developers. Comparing successful projects with less successful ones, the main result that emerged was the importance of establishing a strong customer-orientation for IT projects with high levels of user involvement. Those projects which were more successful were those that were user-led with ITSD staff playing a supportive rather than directive role. It was also noted that for user-led development to be effective, users need to be aware of the need to commit substantial resources of their own. This was not always recognised. Interviews with developers in the ITSD also corroborated the need for intensive user involvement which it was felt could be realised through a prototyping approach. These general methodological observations confirmed the customer-orientated philosophy underpinning the original proposal.

The next step in this technical theme was the mapping out of the current development route practised in the ITSD and the development of an initial Salford Structured Development Methodology (SSDM) drawing together key elements of current and past practices. A high level description of a full life-cycle model has been developed. Further work has examined tool support for the methodology. Oracle's CDM (Customised Development Method) was purchased and evaluated. CDM provides a set of document templates to support a range of life-cycle models.

The next stage in the work was to produce a definitive methodology (SSDM). The aim was to take the initial Salford methodology and to enrich it in various ways:

- Through benchmarking against best practice elsewhere (two organisations were studied: BNFL and Spar Grocery Chain)

- By considering work on a national level oriented towards standard approaches to Rapid Application Development, i.e. the DSDM methodology which has been developed through a consortium of user organisations and technical specialists.

- By taking into account prior work on standards within the ITSD

The definitive SSDM will be defined in the final CAPELLA report. One important result that should be highlighted here is the recommendation to implement Peer Review as a key practice for achieving improved software quality. This will be implemented and evaluated along-side CASE in the third phase of the experiment.

Evaluation

Metrics were seen as falling into two main areas: *Developer oriented* and *Customer oriented*. A set of metrics has been developed in both areas. Regarding developer metrics, the use of Function Points received a detailed examination. It was concluded that function points were both useful and valid way of measuring output, but that it was crucial to apply the analysis at the level of the business function. This was a key result. Choice of software metrics was guided by a *Capability Maturity Model Assessment.*

Cost and benefits

The metrics framework will be deployed in phase 3 enabling the benefits of CASE to be formally evaluated. Preparatory research on costs is also underway; this will take into account both direct costs (e.g. training, support etc.) and indirect costs (e.g. overheads). At a qualitative level, attitudes of the staff towards CASE have been researched through a series of interviews with developers. In general, attitudes to CASE were positive with CASE being seen as providing a number of technical benefits, including the following:

- It facilitates user involvement through prototyping
- It represents the state-of-the-art in software engineering, thus enhancing job satisfaction
- It will greatly improve documentation
- Code generation will improve productivity and reduce defects
- It will promote more re-use

CASE Implementation Strategy

Following the draft strategy, a more detailed analysis has been carried out in relation to the implementation of CASE in order to identify problems with the current approach, the lessons that need to be learned and to consider how benefits might be maximised. Cultural issues have figured largely in this analysis. Resistance to CASE has been attributed to a range of factors, e.g.:

- The degree of new learning that is required

- The reactive nature of current workload
- Low morale and perceived elitism
- Future work mix/Development Strategy (increasing use of Package Solutions)

This analysis has resulted in a set of recommendations and a revised outline implementation strategy, which will focus particularly on the management of change, and on optimising the opportunities provided by CASE for organisational learning.

INTERIM BUSINESS RESULTS

The results of the Project have contributed to the achievement of I.T. Services' Business Objectives in the following areas:

The methodological theme has yielded a draft methodology that addresses the business need to adopt a more customer-focused approach. In particular, the use of prototyping is seen as critical in realising higher levels of user involvement and hence customer orientation and business validity. The introduction of peer review is also seen as providing a decisive first step in instituting a quality management system and establishing a culture of continuous improvement and organisational learning.

The metrics framework is equally important in improving performance. It will provide a basis for setting process improvement targets and monitoring the effectiveness of interventions. Institutionalising the metrics in the main evaluation represents a major move forward in establishing a metrics-based management culture. The current set of metrics have been defined with improved software quality and project management as the key goals. Clear benefits are expected to flow in both these two areas.

From a business perspective CASE is seen as providing an important means of achieving improved performance in four key areas; software quality, IT staff productivity, documentation, and project management. These business benefits are seen as justifying use of CASE for the immediate future although the final decision regarding future deployment will depend on the outcome of the main evaluation.

From a business perspective, the experiment has yielded many important initial results regarding the management of change. Whilst CASE has been successfully used in I.T. Services, there have nonetheless been problems (delays and resistance) and there is no doubt, that the management of the implementation procedure could have been improved. The new implementation strategy is thus of considerable importance from a business perspective, not just for CASE but for the management of any new initiative within the department. Given the turbulence of the ITSD's business environment, the enhancement of its capacity for change is critical.

So far as the project Management role for Software Development Projects within the Service is concerned, several key initiatives have now been implemented including:

- The implementation of rigorous project planning management and review processes supported by standard documentation.

- Increased consultation and participation of other IT disciplines at key project milestones, regarding dependencies beyond immediate software development domain (including data communication, training services etc).

- A Risk Management framework to capture key information and to forecast threats to project deliverables and targets.

Whilst it is difficult to substantiate in a measurable tangible sense, there are at least promising signs of real cultural shifts arising from the project to date which can be categorised as:

- A recognition by a sizeable proportion of the ITSD staff of the need to innovate and exploit technology to aid the development process

- Acceptance of the need for a team approach to development including development staff, customers, suppliers and other stakeholders, from both internal and external agencies.

- Acceptance of the need to be more accountable for quality and performance at individual and team levels through more structured project reviews, peer reviews and use of metrics and measurement techniques.

- A greater acceptance of the role and importance of the customer in the development process.

KEY LESSONS LEARNED

Service Culture

The prevailing Culture within the I.T. Service and how the planned change process involving the CASE approach might be greeted should have been more carefully understood, in order to prepare the way for the experiment. In other words, the determination of the capacity for change is a critical factor in establishing changes in strategic direction, from many important points of view; leadership, management, resources, communication, existing morale, structures and so on.

Strategy Programme

ITSD tends to be an organisation with a strong operational orientation, where strategic planning and management is often perceived as bureaucratic or unnecessary. This might have been recognised more clearly, so that the overall *raison d'être* for change was better communicated, thereby achieving greater acceptance and ownership. Additionally, how the CAPELLA Project fitted with other strategic initiatives within the ITSD was never fully articulated so that an overall coherent strategy programme emerged and it was, therefore, able to be seen within context of the wider goals for the service.

Involvement and Communication

Insufficient attention to communication had led to confusion about the project scope and purpose, roles and responsibilities within the project and its various components, together with its relative priority; leading to insufficient active involvement and ownership, particularly by practitioners.

Project Management

The loss of key staff, including the original project leader dealt a severe blow to the management and continuity of the project. Momentum was lost at a critical formative stage, and this was further exacerbated by the continuing resource demands arising from Year 2000 compliance work. Project management was fundamental and the team benefited from the assignment of a project manager in addition to the overall lead officer, to provide the day-to-day intensive planning and management to various aspects of the project required and a firmer grip on achieving agreed deliverables.

Project Scope

It was during the re-establishment of the project immediately following the key staff losses where the project was revisited in terms of detailed scope. The general philosophy being do few things well, keep them simple, have a keen attention to detail and above all, chunk the work to ensure manageability.

Management and Leadership Perspectives

In many ways, the project is less of a technical one than a managerial one. Furthermore, it is less a project than a whole different strategic agenda for change. It is these management perspectives that were sometimes overlooked in favour of a more limited project view within the technical domain of software engineering. Additionally, a clearer appreciation of the capacities and skills of individual project team members would have been valuable at the outset, perhaps by introducing skills assessment exercises at the start of the programme.

MAIN CONCLUSIONS

The main message so far is that there is a far more fundamental issue at stake here than the costs and benefits of CASE, or indeed how we create opportunities for continuous improvements in the software business function, although clearly those will form a substantial part of the final conclusions. The key to these things lies in the manner in which organisations cope with the change programme affecting them, now and in the future. In other words, for initiatives such as the introduction of CASE to be successfully integrated into the organisation’s business functions and for on-going improvement programmes to be developed and implemented, it is critical that we:

- Understand prevailing cultures within the ITSD in order to facilitate and manage change processes.

- Place the CAPELLA project within context of the overall change programme and communicate the relationship with other strategic initiatives so that an overall coherent strategic programme emerges, achieving understanding and acceptance.

- Ensure that suitable structures and processes are in place to cope with change as the project moves from inception through development and into implementation.

All of these factors are concerned with our capacity to cope with change arising from the project. Such factors are critical to success and need to be in place to underpin the ideas and initiatives flowing from the project. In particular, the involvement of the consultants / researchers has been of major importance to the ITSD. They have made contributions technically, strategically, managerially and on a pragmatic day to day level. They brought experience, expertise and best practice from theoretical as well as practical domains at a time when the ITSD were suffering from a very high level of staff turnover, and competing demands for resources arising from the Year 2000 compliance programme and other matters.

APPENDIX: Author Index