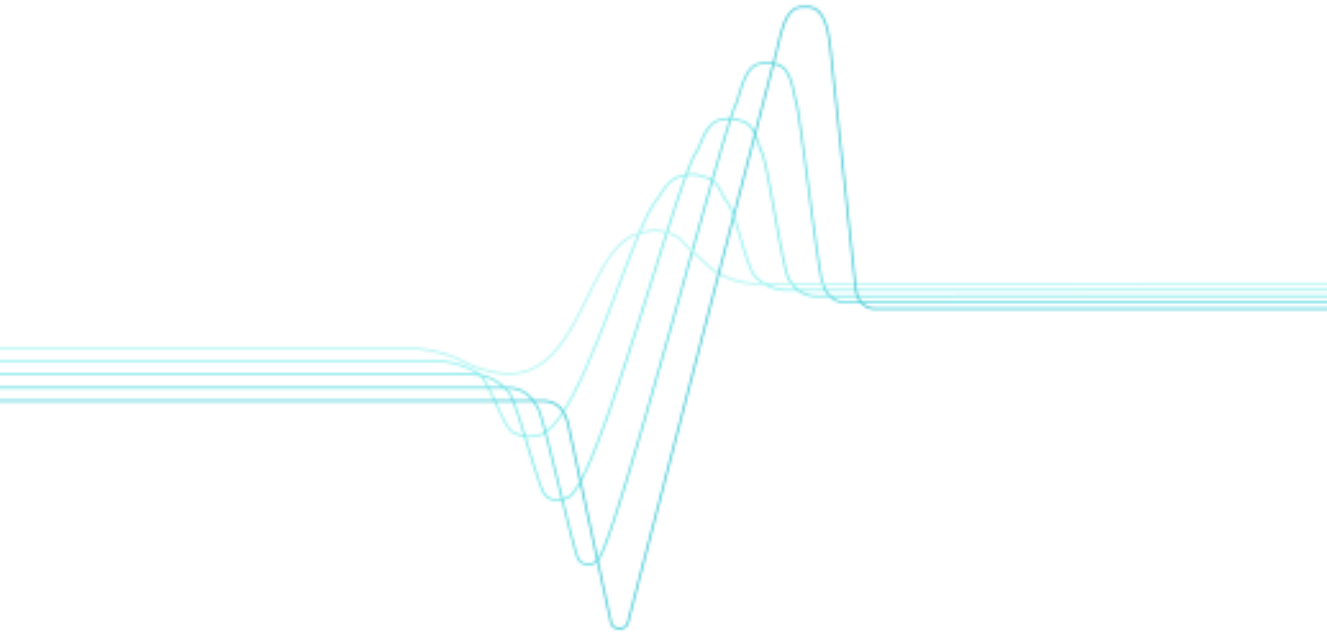
ESPOO 2006 VTT PUBLICATIONS 605

Jukka Kääriäinen

Practical adaptation of configuration management

Three case studies



VTT PUBLICATIONS 605

Practical adaptation of configuration management

Three case studies

Jukka Kääriäinen
VTT

ISBN 951–38–6842–7 (soft back ed.)
ISSN 1235–0621 (soft back ed.)

ISBN 951–38–6843–5 (URL: http://www.vtt.fi/publications/index.jsp)
ISSN 1455–0849 (URL: http://www.vtt.fi/publications/index.jsp)

JULKAISIJA – UTGIVARE – PUBLISHER

VTT, Vuorimiehentie 5, PL 2000, 02044 VTT
puh. vaihde 020 722 111, faksi 020 722 4374

VTT, Bergsmansvägen 5, PB 2000, 02044 VTT
tel. växel 020 722 111, fax 020 722 4374

VTT Technical Research Centre of Finland
Vuorimiehentie 5, P.O.Box 2000, FI–02044 VTT, Finland
phone internat. +358 20 722 111, fax + 358 20 722 4374

VTT, Kaitoväylä 1, PL 1100, 90571 OULU
puh. vaihde 020 722 111, faksi 020 722 2320

VTT, Kaitoväylä 1, PB 1100, 90571 ULEÅBORG
tel. växel 020 722 111, fax 020 722 2320

VTT Technical Research Centre of Finland
Kaitoväylä 1, P.O. Box 1100, FI-90571 OULU, Finland
phone internat. +358 20 722 111, fax +358 20 722 2320

Technical editing Maini Manninen

Otamedia Oy, Espoo 2006

Kääriäinen, Jukka. Practical adaptation of configuration management. Three case studies. Espoo 2006. VTT Publications 605. 71 p. + app. 48 p.

Keywords software engineering, software configuration management, configuration management, embedded systems, agile methods

Abstract

This research studies the adaptation of configuration management. Configuration management (CM) is a support process for product development and it operates in the context of the development project. Several factors, such as the size of the project, distribution, development disciplines, etc. affect the project's CM solution. Nowadays, CM practices inside a project have become an industrial de-facto standard, but the complexity emerges from the modern operational environment of product development. Globalisation, outsourcing, product variation and the amount of SW in modern products have characterized the modern product development. This trend has also affected the CM practices, which need to face these new challenges.

This study defines the initial framework of factors that affect the CM solution. These factors represent the project characteristics that the CM adaptation needs to solve when planning the CM solution for a project. Even though separate factors can be identified, they coexist and therefore the planning of CM has to take these factors into account singly and together.

The framework of factors has been used to characterise three CM adaptation case studies. The case studies represent the two ends of the project types. Case 1 represents a large multisite development project, while cases 2 and 3 represent small SW development projects. The CM practices are considered based on factors in each case and the results are discussed. Furthermore, a cross-case analysis has been carried out to detect and discuss similarities and differences between the cases.

The results indicate that the plan-based CM worked well and provided mechanisms for identifying CM solutions that suited the project context despite

of the project size, although the formality and complexity of the CM solutions varied. Good communication between the product development teams as early as during the CM planning phase was found essential to ensure consistent CM practices.

The study also revealed that inside a project, the CM practices are usually fairly well realised, but the complexity and challenges of CM come from the size of the project (large), work distribution (project hierarchy, multisite development, dependence on third party software) and development disciplines (HW/SW). Especially, the management of interfaces was found crucial in complex development environment. Without strict practices unmanaged interfaces can cause difficult problems in the integration phase.

Preface

This research has been carried out in VTT during the years 2002–2004. The foundation for the research was created in year 2002 in VTT's strategic research programme called UUTE (New software development technologies). The cases have been documented in "Agile software technologies" -project and ITEA-project called MOOSE (Software Engineering MethOdOlogieS for Embedded Systems) during 2002–2004. This thesis was composed in ITEA-projects called AGILE (Agile Software Development of Embedded Systems) and MERLIN (Embedded Systems Engineering in Collaboration) during the years 2005–2006. The work has been originally published as a licentiate thesis at the University of Oulu.

I would like to give my thanks to the following persons that have made this thesis possible. Professor Samuli Saukkonen has been the advisor of my licentiate studies. Professor Oddur Benediktsson and Dr. Jorma Taramaa have been the reviewers of this thesis. They provided valuable comments that improved the quality of this thesis. Professor Pekka Abrahamsson provided valuable comments and guidance throughout the research and writing process.

Furthermore, I would like to give my gratitude to my family for all support and understanding during my studies.

Oulu, May 2006

Jukka Kääriäinen

Contents

List of original publications

The research includes the following original publications:

I Kääriäinen, J. Taramaa, J. & Alenius, J. 2004. Configuration management support for the development of an embedded system: experiences in the telecommunication industry. Tools and methods of competitive engineering. Vol. 2. Millpress. The Fifth International Symposium on Tools and Methods of Competitive Engineering (TMCE 2004). Lausanne, CH, 13–7 April 2004. Pp. 605–616.

II Koskela, J., Kääriäinen, J. & Takalo, J. 2003. Improving software configuration management for extreme programming: a controlled case study. EuroSPI 2003 Proceedings. Verlag der technischen Universität Graz, European Software Process Improvement, EuroSPI'2003. Graz, Austria, 10–12 Dec. 2003.

III Kääriäinen, J., Koskela, J., Takalo, J., Abrahamsson, P. & Kolehmainen, K. 2003. Supporting requirements engineering in extreme programming: managing user stories. Proceedings of the ICSSEA 2003, 16th International Conference, Software Systems Engineering and their Applications, Vol. 4. Paris, FR, 2–4 Dec. 2003, ICSSEA.

IV Kääriäinen, J. Koskela, J., Abrahamsson, P. & Takalo, J. 2004. Improving requirements management in extreme programming with tool support – an improvement attempt that failed. 30th Euromicro Conference, EUROMICRO 2004, Rennes, 31 Aug.–3 Sept. 2004. IEEE Computer Society. Pp. 342–351.

List of acronyms

ASIC	Application Specific Integrated Circuit
CCB	Change Control Board
CM	Configuration Management
CMO	Configuration Management Officer
CVS	Concurrent Versions System
DM	Document Management
DSP	Digital Signal Processing
ERP	Enterprise Resource Planning
FCA	Functional Configuration Audit
FPGA	Field Programmable Gate Array
HW	Hardware
O&M	Operation and Maintenance
PCA	Physical Configuration Audit
PDM	Product Data Management
PLM	Product Lifecycle Management
RM	Requirements Management
SCM	Software Configuration Management
STD	State Transition Diagram
SW	Software
VTT	Technical Research Centre of Finland
XP	eXtreme Programming

1. Introduction

The use of software has increased in many different product types. Crnkovic et al. (2003) present that in 1990 the development costs of software (SW) were one third of the total development costs of industrial robots while now it is two thirds. ITEA (2005) presents that investments for SW research and development will increase dramatically in the near future (Figure 1).

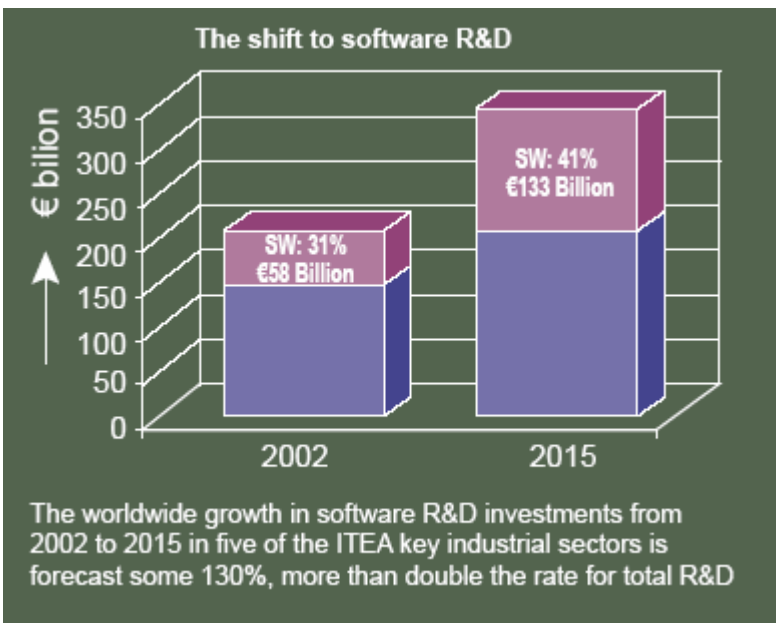


Figure 1. Forecast for the worldwide growth of R&D investment (ITEA 2005).

The ability to produce quality products on time and at competitive costs is crucial for any industrial organisation. Nowadays, also the needs of the customers have diverged. The customer specific product variation is often implemented using SW rather than hardware (HW) and these customer specific versions of the products need to be maintained.

The discipline that keeps the evolving product under control is called Configuration Management (CM). It is a well-known concept in software engineering and it has been widely discussed in literature and articles (for example in Moreira (2004), Jonassen Hass (2003), Leon (2000), Estublier (2000), Dart (1996), Buckley (1996), Berlack (1992), Bersoff et al. (1980),

ISO/IEC 10007 (1995), ISO/IEC 12207 (1995), IEEE Std-828 (1998) and IEEE Std-1042 (1987)). Configuration management has been identified as an essential element in increasing product quality, development efficiency and enterprise profitability (Schamp & Owens 1997).

Practical CM solutions need to be planned in the context of a product development project. Factors, such as the size and the type of the product being developed, size of the project, project distribution, etc. shape the CM practices of the project. Under the different factors, CM needs to address various challenges. For example, in a multisite development environment, work allocation and information sharing are crucial in controlling the development activities. In literature, the concept of configuration management has been examined in various contexts, such as:

- Multitechnology products (SW/HW) e.g. in Taramaa (1998), Lyon (1999), Buckley (1996) and Crnkovic et al. (2003).
- Agile development e.g. in Christensen (2001), Paulk (2001), Jonassen Hass (2003) and Berczuk and Appelton (2003).
- Multisite development e.g. in Ebert and De Neve (2001), Battin et al. (2001), Rahikkala (2000) and Jonassen Hass (2003).

SW development of significant products is a very complex undertaking. The complexity of CM is a result of its context dependent nature. CM is a support process that is adapted into the product development environment. Although the basic elements of CM are the same, the interaction of several factors cause that the practical implementation of CM varies. We are also living in a changing world requiring more flexibility and abilities to adapt to new development approaches to survive in the global competition. One example of this is the shift from the traditional in-house product development to the collaborative development environment with the use of subcontracting and SW components. The changing development environment demands that CM face these new requirements whenever they emerge.

1.1 Scope of the Research and the Research Problem

This research focuses on the context dependent nature of CM. It considers the planning process and activities of CM and examines what kind of factors influence the CM solutions in the SW development. As SW is an important part of any modern electronics product, the connections to the HW development will also be considered.

The goal of the research is to gain a better understanding of the factors that shape the CM solutions of an organisation. The research will identify the initial set of factors forming a basis for further research. These factors will be used to analyse case studies that are from the fields of embedded system development, information system development and mobile SW development.

Based on the discussion above, we state the research questions as follows:

- What is the basic mechanism for adapting configuration management?
- What kind of factors influence the configuration management solutions?
- How these factors affect the CM planning and the practical CM solutions?

1.2 Structure of the Research

Section 1 introduces the structure of the study and discusses the research problem and research methods used in this study.

Section 2 presents an overview and concepts of configuration management. Then, it introduces the principal elements of CM, including planning, identification, control, status accounting and auditing. Furthermore, it also considers the role of CM in the system life cycle and the automation of CM.

Section 3 presents a framework that describes the factors that affect the CM process instantiation. This framework is used to analyse the CM solutions in three case studies in section 4. First, section 4 introduces the cases based on the factors of the framework. Next the section presents how these factors map with

CM solutions in the cases. Finally, the section presents a cross-case analysis over the cases and discusses the similarities and differences of the cases.

Section 5 introduces papers that have been included into this research. Section 6 discusses the results gained from the analysis, answers the research questions and states future research needs.

1.3 Research Methods

The selection and adaptation of the CM solution for a project is a complex where several factors affect the selected solution. The concepts and factors related to the adaptation of CM were considered based on literature study. This study provides a conceptual basis for the research and resulted in an overall framework of the factors that affect the adaptation of CM. These factors reveal the context-dependent nature of CM.

The adaptation of CM was studied empirically through three case studies as follows (see section 5 for more information on the papers behind the cases):

- Case 1 considers observations while adapting CM for a project developing digital signal processing (DSP) SW and HW related DSP SW in a distributed development environment containing HW/SW codesign.
- Case 2 considers observations while adapting CM for a project (named eXpert) that used agile development principles for producing a SW system for managing the research data and documents.
- Case 3 considers observations while adapting CM and improving RM for a project (named zOmbie) that used agile development principles for developing a financial sector mobile SW.

Yin (1994) defines a case study a method that examines a phenomenon within its real-life context. This method is suitable because it tries to produce intensive and detailed information on the context-dependent cases. The method also provides room for the diversity and complexity of a phenomenon. This is essential when practical solutions are the results of the interaction of many factors. Action research (e.g. in Hult and Lennung (1980) and Susman and Evered (1978)) is a

method that is typically used by practitioners who analyze the data to improve their own practice. It is used to improve the quality of an organization and its performance in the active interaction of research and practice. It aims at better understanding of a theory while improving the state of the practice in an organisation.

The context of each case (e.g. size of the project, distribution, methods used) was introduced using the framework defined in this research. Based on Yin (1994), the study can be based on one or several cases. In this research, first the configuration management solutions in each case were introduced and discussed. Then, these three case studies were examined using a cross case analysis to discuss similarities and differences between the cases.

2. Configuration Management

The purpose of this section is to provide an introduction for configuration management. The section introduces the concepts, activities and tool issues of configuration management.

2.1 Overview

The roots of CM are in the defence industry environment as a discipline to resolve problems with poor product quality, parts ordering, and parts not fitting, which were leading to high cost overruns (Berlack 1992). Control and communication problems lead to the first standard for CM, called AFSCM 375-1 (Berlack 1992, Leon 2000). At the beginning, the focus was on the CM of hardware oriented products. The need for the management of software artefacts became topical as software engineering industry emerged. According to Estublier et al. (2005) software CM (SCM) emerged as a separate discipline in the 1970s, with the advent of tools such as SCCS, RCS and Make. The traditional products that were composed based on mechanical and electronics components included more and more software. Nowadays, also software development as an engineering activity is more complex. It ties up more developers from different cultural backgrounds, as globalisation removes national borders. Furthermore, the news that a product is bad and has a bug can spread very fast (e.g. newsgroups), which forces a company to provide the fixes and patches very quickly to save face and prevent its market share from dropping (Leon 2000).

The term *configuration management* is originally directed for the management of hardware oriented systems while the term *software configuration management* can be defined as “configuration management tailored to systems, or portions of systems, that are comprised predominantly of software” (Bersoff et al. 1980). Software configuration management and hardware configuration management are quite similar. They both aim at the same objectives. However, Tichy (1988) presents that SCM differs from CM in the two ways: software is easier to change and SCM is easier to automate.

The evolution of software configuration management has been driven by the research community (Crnkovic et al. 2001). It has been identified as a mature and one of the most successful branches of software engineering (Estublier et al. 2002). In this study, CM is treated especially from the point of view of software development.

2.2 Concepts

This section explains the key concepts that are used in the CM literature and that will be used in the following sections of the study.

Configuration management is defined by the IEEE Std 610.12 (1990) as a discipline applying technical and administrative direction and surveillance to identify and document the functional and physical characteristics of a configuration item, control changes to those characteristics, record and report change processing and implementation status, and verify compliance with specified requirements. In short, CM is a discipline controlling the consistency between the parts of an entire system. Standards, such as ISO/IEC 10007 (1995) and ISO/IEC 12207 (1995) introduce it as a support process for product development.

Configuration item is treated as a single entity in the CM process. Item types that are subject to CM may be, for instance, in-house developed or purchased from a vendor (Buckley 1996). Further, these items can be deliverable items under the contract or used to produce the deliverable item (Buckley 1996). The selection of configuration items is important because different types of configuration items have different needs for control. Buckley (1996) divides the software item selection into two parts: the selection of software categories (e.g. product software, vendor-provided software and test software) and the identification of item types in each category (e.g. source files, documents and executables). The role of the item selection is to determine these classes. A software category that is often forgotten is a vendor-provided software that needs special attention, for instance, from the change management point of view. IEEE Std-828 (1998) highlights the importance of two software categories. First, subcontractor/vendor control needs special attention because of the organisational and legal relationships. According to the standard, there has to be practices for how the software will be received, tested, and placed under CM; how changes to the

supplier's software are to be processed; and whether and how the supplier will participate in the project's change management process. Second, the external items to which the project software interfaces need to be identified and managed.

Term *version* is used to describe the evolution of the item (item's versions) and a term product *variant* refers to the development of families of related products. *Check-out* is the process of copying the item from the CM tool to the user's working area for modification. On the other hand, *check-in* is the process of moving the configuration items into a CM tool. This operation produces a new version of the item. The version control mechanisms can be divided into: *pessimistic* and *optimistic* version control (Mens 2002):

- With *pessimistic version control*, all participants work on the same set of software artefacts and parallel editing of the same artefact is prevented by *locking* (when checking out an artefact for modifications).
- With *optimistic version control*, each developer can work on a personal copy of a software artefact (multiple check-outs). This requires mechanisms (merge) to integrate the personal copies when checking in the code back into the version control.

Basically, versions are described as the linear set of developed items (v1 => v2 => v3). Linear development is not always possible and thus we need the concept *branch* (Leon 2000). Branches may be needed for the bug-fixing of the old version of the product or for the parallel development of a single file. According to Leon (2000), branches are deviations from the main development line for the item and they can also be extended from the existing branch. The term *merge* expresses the incorporation of parallel changes with the main development line.

A software development process transforms the description of the SW system from abstract to concrete, typically from requirements specification via design to implementation. The different configurations (a collection of configuration items) need to be managed during the development process. However, all the versions are not equally important. The configuration of the software at a discrete point in time is known as *a baseline* (Leon 2000). The baseline is the cornerstone of CM in general. The baseline serves as a reference point for the next development activity (Figure 2).

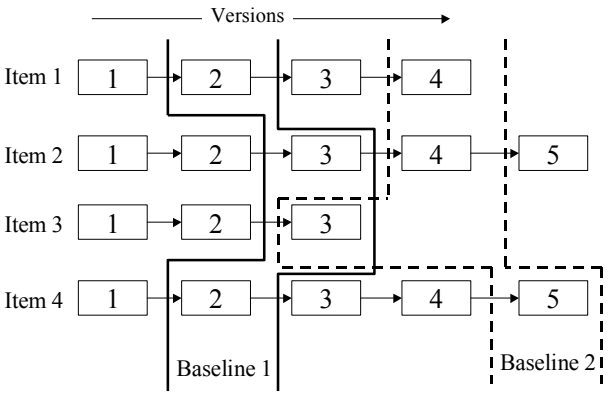


Figure 2. Two baselines from the same items.

A series of different baselines are established to permit an ordered flow of development work (Buckley 1996). A baseline is a specification or product that has been formally reviewed and agreed on, which thereafter serves as the basis for further development and which can only be changed through formal change control procedures. The baseline types that are typically presented in the CM literature are *functional baseline*, *allocated baseline* and *product baseline* (see the section 2.4).

Workspace is a development area where a set of items are collected together for a particular purpose. Workspaces are needed, for example, for development work and for builds and releases. *SW build* is the process of generating an executable, testable system from the source code. From the CM point of view, the build process concentrates on configuration selection problems. According to Abran and Moore (2004):

Software building is the activity of combining the correct versions of software configuration items, using the appropriate configuration data, into an executable program for delivery to a customer or other recipient, such as the testing activity. For systems with hardware or firmware, the executable program is delivered to the system-building activity.

A build process needs information on items (selection of items) and item versions (version selection) to be included into a certain configuration (Whitgift 1991). In addition to this, there must be a construction model (i.e. configuration

data) that is usually described as a makefile. In practice, the selection of items and versions can be done using the CM tools' functionalities (e.g. a workspace populated with right items and item versions).

The term *release* can be considered a CM action whereby a particular version of software is made available for a specific purpose (e.g. released for test) (Buckley 1996). The release should contain all required information to enable the traceability of the configuration of the product. Therefore, in addition to the product configuration, there is also a need to store information on the environment where the product was produced, such as the operating system, compile and link parameters, etc. (Leon 2000).

2.3 The Principal Elements

The CM process contains the basic CM activities and CM planning (e.g. in ISO/IEC 12207 1995, Buckley 1996, Taramaa 1998). Traditionally CM activities have been divided into configuration identification, configuration control, configuration status accounting, and configuration audit (Figure 3) (ISO/IEC 10007 1995, Bersoff et al. 1979). These elements will be introduced in the next sub-sections.

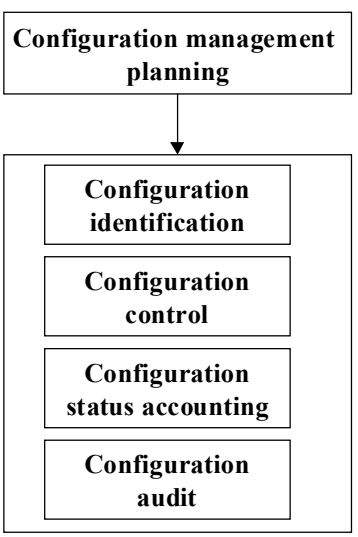


Figure 3. The elements of configuration management.

2.3.1 Configuration Management Planning

Configuration management planning provides mechanisms for planning and documenting the CM solution for a project. A document that describes this information is called a *configuration management plan*. It describes who is going to do what, when, where and how related to CM (Buckley 1996). The elaborateness of a CM plan is situation-dependent. Buckley (1996) presents that all can be placed into a CM plan or to the relegate details. In the former case, changes to details cause changes to the plan itself. In the latter case, the CM plan can be smaller and more useful as a management document. Buckley (1996) states that the latter solution is the better approach in a well-established CM culture.

Literature (e.g. Buckley 1996, Berlack 1992 and Lyon 1999) and standards (IEEE Std-828 1998) assist the CM planning by providing pre-structured templates for documenting the CM responsibilities and practices. Especially IEEE Std-828 (1998) has become a widely accepted template for the CM planning. IEEE Std-828 (1998) presents the following classes of information:

- *Introduction* that describes the purpose, scope of application, key terms, and references
- *CM management* that identifies the responsibilities and authorities for accomplishing the planned activities
- *CM activities* that identify all activities to be performed in applying to the project
- *CM schedules* that identify the required coordination of the CM activities with the other activities in the project
- *CM resources* that identify the tools and physical and human resources required for execution of the plan
- *CM plan maintenance* that identify how the plan will be kept current while in effect.

The main roles that are involved in the CM are the *CM team* and the *Change Control Board (CCB)*. The CM team (Leon 2000) is an organizational support function. In practice, it can be a single person or a full team, depending on the situation (e.g. the size of a project). According to Leon (2000), a project's CM

team contains a Configuration Management Officer (CMO), and depending on the project, other technical (e.g. a person who is responsible for release and build activities) and administrative (e.g. secretary) members assigned by the CMO, for example according to the basic CM activities. The Change Control Board (CCB) contains persons who are qualified to comprehend the scope and impact of a change (Moreina 2004). Buckley (1996) states that the CCBs can also form a hierarchy where the main CCB will delegate the authority of approving specified classes of auctions to the chairs of lower-level boards. In practice, the CCB can be a single person or a full team, depending on the situation (e.g. the size of a project).

IEEE Std-828 (1998) states, that *a successful CM plan reflects its project environment. It should be written in terms familiar to its users and should be consistent with the development and procurement processes of the project.* So each software project is unique with its own characteristics, such as product type, size, methods and tools. While no two projects are exactly the same, the CM requirements between different projects also vary (Lyon 1999, Leon 2000, Whitgift 1991). For example, IEEE Std-1042 (1987) states that the size, complexity and criticality of the software system being managed affect the project's CM practices. Whitgift (1991) lists the following factors that affect the CM solution, among others: the size of the project, project distribution, dependence on third party software, the item types being developed, and the number of clients. In addition, the software and hardware development have special needs for the CM procedures (Buckley 1996). Factors affecting the CM solutions will be considered in section 3.

Stevens et al. (1998) argue that in complex systems separate development projects for each subsystem are usually needed. Each project can lead to the creation of further development projects at a level below. Abran and Moore (2004) present that software CM activities take place in parallel with hardware CM activities when software is developed as a part of a larger system containing hardware. They further mention that the software CM activities must be consistent with the system CM activities. On the other hand, Lyon (1999) stresses the importance of good communication between HW and SW elements of CM during the CM planning. The case of multi-level contracts is similar. The main contractor's CM plan is the main CM plan, and the subcontractors will prepare their own plans or include their plans in the main CM plan (ISO/IEC

10007, 1995). ISO/IEC 10007 (1995) further states that these plans, indicating different levels of configuration management, need to be compatible with each other. The configuration management can also be considered based on target levels as described in Moreira (2004). Moreira (2004) defines these levels as an *organisation,* an *application* and a *project* and examines configuration management tasks on each target level.

The differences between projects do not mean that the planning of a CM solution should always start from "scratch". Existing generic CM templates, such as in IEEE Std-828 (1998), provide a good starting point for systematising CM in a company. Also, experiences from previous project-specific CM plans can be utilised for defining the CM solutions for new projects. A company without any CM can start with project-specific CM planning and then, later on according to experiences, start to prepare a company-specific general plan, which then can be tailored for the projects (Buckley 1996). This will facilitate the definition of a project-specific CM plan, because a generic CM plan provides a "tried and tested", pre-defined structure and information for the creation of the project's CM plan. This approach provides the possibility of reusing the CM practices between projects and assists towards more consistent CM in a company. However, IEEE Std-1042 (1987) states that even if CM is applied as a corporate policy, it should be re-examined each time it is applied in a project to ensure its applicability.

CM activities planned for a project have traditionally been divided into four classes. Next, these classes will be discussed in detail.

2.3.2 Configuration Identification

ISO 10007 (1995) describes *configuration identification* as activities comprising determination of the product structure, selection of configuration items, documenting the configuration item's physical and functional characteristics including interfaces and subsequent changes, and allocating identification characters (IDs) or numbers to the configuration items and their documents.

Configuration item selection identifies the categories of items that will be subject to CM. The selection of the configuration items is important because

different types of configuration items have different needs for control. Furthermore, each configuration item needs to be uniquely *identified* so that it can be distinguished from other items and from the different versions of the same item. This requires mechanisms for naming and versioning these items (IEEE Std-828 1998). Item identification as an activity is an essential prerequisite for other CM activities presented in sections 2.3.3, 2.3.4 and 2.3.5. The configuration of software at a discrete point in time is known as a baseline (Leon 2000). Baselines that will be produced should be identified. IEEE Std-828 (1998) presents that the following must be defined for a project:

- the event that creates the baseline;
- the items that are to be controlled in the baseline;
- the procedures used to establish and change the baseline;
- the authority required to approve changes to the approved baselined documents.

Further, configuration items need to be *acquired* under the control of the configuration library. These practices describe where configuration items are physically stored, by whom, when and how. The practices should also describe the access control and item retrieve issues. In practice, this means a set-up and guidance for the use of the CM tool.

2.3.3 Configuration Control

ISO 10007 (1995) describes *configuration control* as activities comprising the control of changes to a configuration item after the formal establishment of its configuration documents. This means the management of changes to the baselined configuration items.

There are various descriptions and models for the phases of change management in literature (e.g. presented in Mäkäräinen 2000). However, usually certain high-level steps are common for the change management models regardless of the model. The change management contains elements for change identification/documentation (change request), evaluation, decision and implementation (Figure 4). The objective of systematic change management is to ensure that similar information

is collected for each proposed change and that overall judgements are made of the cost and benefits of the proposed change, as well as after the change, the new baseline is produced.

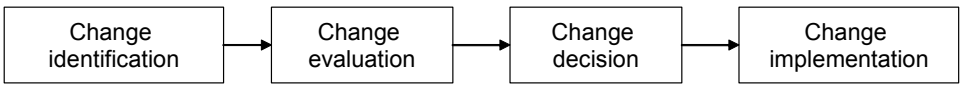


Figure 4. Change management process.

Change identification and documentation is used for collecting information needed for characterising the proposed change and for providing sufficient information for analysing/evaluating the change. The responsibility for the change evaluation and decision should be explicit (e.g. defined CCB). Change implementation realises the actual change to the configuration items. After the change has been implemented and verified, the change history (Leon 2000) will be recorded describing the "life" of the change including information, such as originator, analysis done date, approving authority, approval date, etc. According to Leon (2000), this information can be automatically recorded as and when the events are happening when using the CM tools.

In practice, there are different kinds of change requests of which some are more urgent than the others. A change can be, for instance, a result of a fault, enhancement or changed requirement. The most significant distinction between requests is whether the request represents an enhancement or reports a fault (also known as an incident, defect or bug) (Whitgift 1991). Usually, a fault gets immediate attention because it is a result of a problem and the problem needs to be fixed. Leon (2000) presents the following process (Figure 5) for managing a fault (a defect). There are also activities for defect prevention including causal analysis and the storage of fault information to the knowledge database. Leon (2000) states that when a fault is reported, the knowledge database can be searched for similar problems and if one exists, the solution for the previous bug will help resolve the underlying fault faster.

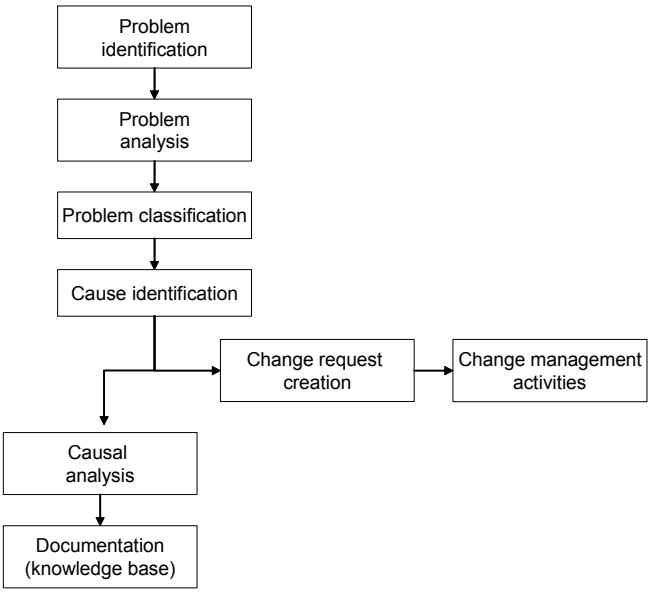


Figure 5. Problem reporting and tracking process (Leon 2000).

2.3.4 Configuration Status Accounting

ISO 10007 (1995) describes *configuration status accounting* as formalized recording and reporting of the established configuration documents, the status of the proposed changes and the status of the implementation of the approved changes.

According to IEEE Std-828 (1998), the following things should be considered for the configuration status accounting: what data elements are to be tracked and reported for baselines and changes; what types of status accounting reports are to be generated and their frequency; how information is to be collected, stored, processed, and reported; how access to the status data is to be controlled.

The role of the status accounting is to gather, process and present configuration item related information for an organisation. Stakeholders who need the information on the configuration items for their daily work are, for example, the project manager, CCB, CM officer and audit team. For example, the status accounting helps to monitor the progress of the project and provides the reports

of configuration items for the maintenance to help resolve problems faster. Buckley (1996) divides the types of reports to periodic reports and "on-demand" reports, and present examples of these reports. The processing of periodic reports, such as "list of problem reports" or "change implementation status", should be automatic. On the other hand, "on-demand" reports are needed more sporadically, e.g. "where-used" report which, for maintenance reasons, indicates all those product releases where a certain component has been used. The CM tools usually provide reporting facilities that can be utilised for generating automatic reports that contain the required information on the configuration items.

2.3.5 Configuration Auditing

ISO 10007 (1995) describes *configuration auditing* as an examination to determine whether a configuration item conforms to its configuration documents. In practice, the configuration audits are performed before a configuration is released to a client.

Responsibilities and procedures for audits should be identified. Literature further divides audit activity to *functional, physical* and so called *in-process* configuration audit (e.g. in Leon 2000, Buckley 1996). Functional configuration audit (FCA) verifies that a configuration item's actual performance agrees with its software requirements as stated in requirements specification (Berlack 1992). This happens at the end of the development cycle and requires that tests are completed and reported. The physical configuration audit (PCA) determines that the design and product specifications and referenced documents represent the software that was coded and tested for specified configuration items (Berlack 1992). The overall role of the functional and physical configuration audit for the SW development is well summarised in Leon (2000):

Whereas the functional configuration audits authenticate that the software performs in accordance with the requirements and as stated in the documentation, the physical configuration audit authenticates that the components to be delivered actually exist and that they are complete in all respect and contain all of the required items.

The product baseline is established when the functional and physical configuration audits are successfully performed. Leon (2000) states that the audits can be done also at the end of a phase in the development life cycle. This has also been stated in Bersoff et al. (1980). Bersoff et al. (1980) also present checklists for different types of baseline audits. However, the reality is that usually it is only conducted before a system release (release that will go to the customer) (Leon 2000). The in-process audit (Buckley 1996) (also known as the CM system audit or the system configuration audit) is performed to determine whether the configuration management process established in an organization is being followed and to ascertain what needs to be improved.

Configuration audits are usually assisted using checklists and audit procedures. The audit checklists contain issues or questions which will be discussed in the audit session and approval is indicated for each issue as well as possible corrective actions are identified. Different checklists for audits are presented in, for example, Berlack (1992), Bersoff et al. (1980), and Moreira (2004).

2.4 Configuration Management on System Life Cycle

Product development is a process transforming an abstract system description to a concrete product. This evolution can be presented as a system life cycle. The system life cycle represents those stages through which the systems pass as they mature (Bersoff et al. 1980). Therefore, the life cycle stages provide the framework for the identification of milestones and the ultimate control of the system throughout its life (Bersoff et al. 1980). Configuration management is a support process for the product development. It captures, manages and shares artefacts that are produced and applied during the product development process. Therefore, it is necessary to examine, how CM maps with the typical SW development life cycle phases to understand its relations with the SW development activities. The various artefacts are managed under CM in different stages of the life cycle. Furthermore, the various CM activities are needed in different stages of the life cycle. The examples on how the stages of the life cycle, development artefacts, and CM activities are related, are presented, for instance, in Bersoff et al. (1980), and Leon (2000).

Bersoff et al. (1980) define that *life cycle management is principally baseline management*. The baselines provide reference points for the next development steps and traceability. The main baseline types typically presented in literature are *functional baseline, allocated baseline* and *product baseline*. The functional baseline is established after the system specification phase. This baseline describes the functions that the system, acting as a whole, is to perform (Buckley 1996). The allocated baseline is established after allocating the system requirements to the hardware and software systems. The allocated baseline contains specifications for the hardware and software systems and interface requirements documents (Buckley 1996). In complex systems, requirements are allocated first on the sub-system level before the allocation to the hardware and software requirements. The baselines that indicate the sub-system levels before HW/SW allocation are called *subsystem allocated baselines* (Buckley 1996). The product baseline represents the technical and support documentation established after the successful completion of the functional and physical configuration audit (Leon 2000). The product baseline is usually formally delivered to the customer. In addition to the formal baselines, there are *developmental configurations* that define the evolving configurations during the period between the allocated and the product baselines (Berlack 1992). Developmental configurations are used to achieve control of the internal development activities (e.g. tests, reviews).

2.5 Tools

Without automation CM is a manual and time-consuming activity. However, nowadays organisations use CM tools for various CM tasks. The role of the CM tools is to support and automate the CM tasks and to provide help for the developers. Leon (2000) states that the CM tools do not solve configuration management problems, but they can be one step towards a more effective CM. There are dozens of CM tools available, containing features depending on their backgrounds and purposes. Basically, the tools can be divided into two main classes: tools that contain just the sub-set of the CM support and tools that contain an extensive CM support (Leon (2000) uses here a term “full-fledged tool”). Tools that belong to the former class may lack, for instance, change management facilities. The examples of tools that contain just a sub-set of the CM functionality are, for example, CVS (Concurrent Versions System) and

Microsoft SourceSafe. Examples of tools that belong to the latter class are Dimensions, ClearCase and Synergy. Estublier (2000) divides the basic functionality of the CM tools to three main classes: *repository for components*, *help for engineers' usual activities* and *process control and support*.

Component repository provides a basic functionality for storing and sharing the product related information, e.g. version management and access control. One characteristic of the software CM tools is the ability to handle complex version structures. This means that the version management also contains support for version tree (branching), where any version can be used to create a new version.

Engineers' support contains support for workspaces, which are views to a certain set of versioned files for a particular purpose, e.g., development, bug-fixing, or testing. The merge facilities are needed to support the resynchronization of parallel changes during the cooperative work. In addition, the SW building facilities support the combination of source files into an executable system.

Process support provides means to support a company's pre-defined processes (e.g., product development process, change process, etc.). There are basically two techniques that can be used to provide support for the processes: State Transition Diagrams (STD) and Activity Centered Modelling (Estublier 2000). In practice, STD based solutions provide a possibility to define the states and transitions between the states for an item type to support operation according to the company's processes. Therefore, it describes the legal way how an item can evolve (Estublier 2000). In Activity Centered Modelling the activity plays the central role, and models express the data and control flow between activities (Estublier 2000). This type of modelling emphasises the work (or tasks) that needs to be done. STD-based solutions are common in SCM tools. However, both techniques are needed for efficient process support, but the integration is not easy (Estublier 2000).

2.6 Summary

Configuration management is a support process for a product development. It aims at controlling the consistency between the parts of an entire system. CM is divided into the CM planning and CM activities. The CM activities comprise:

- *configuration identification*, used to identify the configuration items that are subject to CM. It is a prerequisite for effective CM.
- *configuration control*, an activity for controlling changes to identified and approved configuration items.
- *configuration status accounting*, an activity for collecting and reporting information on configuration items and changes.
- *configuration audit*, that is an activity for ensuring that the product produced corresponds what has been specified and that all needed artefacts have been produced.

Basically, CM is a discipline that provides control for the product's life cycle management. It captures, manages and shares artefacts that are produced and applied during the product development process. In the past, this was a manual activity that tied up lots of resources and was time-consuming. However, nowadays there are dozens of sophisticated CM tools that are capable of automating the CM tasks substantially.

3. Adapting Configuration Management in Practice

CM is a support process for product development. Therefore, the context of the development project sets various conditions and constraints for it. This section constructs the framework that describes factors affecting the CM solution.

3.1 Factors Affecting the Configuration Management Solution

This section discusses the initial set of factors that affect the CM solutions in an organisation. The factors have been collected from literature.

3.1.1 Size of the Project

CM is necessary in spite of the project size (Leon 2000). It is needed in small and big projects. However, the formality and scale of the CM varies. According to Leon (2000), in a small project, a single person can be responsible for CM. Whitgift (1991) argues that in a small project it can even be, for example, the project manager who carries out CM as a part time job. On the other hand, in moderately large projects (several hundred people) there can be a hierarchy of CCBs and a team responsible for the software library.

3.1.2 Product Type

The type of the product affects the CM solution. Product type can be considered from different viewpoints. For instance, reliability (safety criticality), is product embedded or not, complexity, etc. In this thesis the focus is on criticality. Jonassen Hass (2003) gives an example that divides the product types into different classes according to criticality. She mentions that it is difficult to provide unambiguous rules for CM for different classes of products, but the need for formality and automation increases with the level of the criticality. For example, in a safety critical product that may cause many people to be killed, the need of a definitely careful CM is high.

3.1.3 Project Hierarchy (Distribution)

If the development is organised around one project, the CM activities are quite straightforward. In complex systems, separate development projects for each subsystem are usually needed (Stevens et al. 1998). Each project can lead to the creation of further development projects at the level below. Whitgift (1991) considers this under the term "Project distribution" where a large project is divided into several parts. In a distributed project environment, Whitgift (1991) states that the CM plan must describe the effective CM procedures for each part of the distributed project, how the interfaces between the parts are defined and controlled, as well as how configurations of the complete system are defined and built.

The CM planning should ensure that the CM practices are defined in a unified manner to avoid inconsistencies between the projects and to prevent problems in the integration phase. Therefore, the CM plans indicating different levels of CM need to be compatible with each other (e.g. in Abran and Moore 2004).

3.1.4 Multisite Development

Multisite development means that the development work has been spread to several geographically distributed sites. This kind of an environment requires means to organise and control the development work from the CM point of view. CM needs to address the challenges of networking, communication, security, and concurrency management (Leon 2000). Jonassen Hass (2003) describes that in practice, multisite development often means that, for example:

- the information needs to be doubled and synchronised;
- responsibility for items needs to be clear;
- need for formality and automation increases;
- all sites must follow the same process (if not, the differences must be made clear).

Ebert and De Neve (2001) highlight some best practices to support the global SW development. Related to CM, they advise to rigorously enforce CM and build management rules (such as branching, merging, and synchronisation

frequency). They also highlight the importance of the tool support in this kind of an environment. Battin et al. (2001) highlight the global development issues that Motorola faced when developing software in a global environment. Further, they describe approaches to resolve these issues. They indicate that configuration management is challenging in a globally distributed environment and a solution for this is a common CM tool with multisite replication and a centralised bug repository. Rahikkala (2000) has considered this problem from virtual software corporation point of view and presents that most of the SCM challenges are related to the SCM teamwork process (consisting e.g. security, communication, infrastructure, etc. issues).

3.1.5 Development Disciplines

The development of the modern complex systems requires the cooperation of different technologies (engineering disciplines). The product development might be divided into discipline specific development processes such as software development, electronics design and mechanics design. Each of these development processes produce product related data and there are interconnections between this information. For example, changes to SW might cause changes to HW and vice versa. The management of the product data is challenging in this kind of an environment. Crnkovic et al. (2003) describe the typical problems in a multidiscipline development environment:

- The users of both domains (i.e. HW and SW) believe that the (product information management) system they use can manage all situations and they do not understand the specific requirements of the other domain.
- Different terminology is used for the same concepts and the same terminology is used for different behaviour.
- The users of the HW and SW product information management systems are often located in different departments.

Jonassen Hass (2003) underlines that consistent identification practices and careful change management are needed for those items (e.g. related SW and HW items) that are naturally related. On the other hand, Lyon (1999) stresses the importance of good communication between HW and SW elements of CM during the CM planning.

3.1.6 Development Models

CM is a support process for product development. Therefore, it operates in the context of the development process with the elements the development process produces. There are different methods and approaches for a product development that have certain requirements for the CM practices (Jonassen Hass 2003). The characteristics of these methods and approaches should be taken into account when considering the CM support for a project.

The traditional sequential development model or its extensions, the so-called V- and W-models, emphasise the "doing it right the first time" -viewpoint (Jonassen Hass 2003). On the other hand, the new development models, called agile-methods, emphasise communication and iterative development and welcome change (e.g. in Abrahamsson et al. 2002). The characteristics of the development models shape the CM solutions. For example, an iterative development (typical for agile methods and for HW/SW codesign (Ronkainen and Abrahamsson 2003) requires a good version management and baselining to ensure sufficient traceability.

3.1.7 Dependence on Third Party Software

Projects use different kinds of third party software, such as compilers, tools, software components, operating systems, etc. Whitgift (1991) states that there must be practices on how the third party software is to be specified, delivered, accepted, and changed. Berczuk and Appelton (2003) say that when using third party components the challenge is to associate the versions of these components with your product. And this is because the vendor release cycles are probably different from your release cycles. It is important to ensure that all development teams access the correct versions of the components to avoid problems in integration. Furthermore, adaptations might be needed to customize the third party components to fit your particular needs (Berczuk and Appelton 2003).

3.1.8 Maintenance and Multivariants

If the system has only one customer, many CM activities are simpler (Whitgift 1991). In the case of several customers, the system might have to run in several environments with the customer specific features. On the other hand, bug fixing, new features and enhancements might be needed for maintenance reasons producing the new versions of the product. Maintenance and multivariant issues should be considered in the CM plan (Whitgift 1991):

- How are the permanent variants of the systems managed?
- How are the releases of the system to different customers built and recorded?
- How are old versions of the systems which are still in operational use maintained, and for how long?

According to Jonassen Hass (2003), the CM of multivariants needs, for example, clear naming principles for variants and a change control that is performed with special care.

3.1.9 Item Types

The product development process refines the product description from abstract to concrete through steps of the requirements specification, design, code, integration, and testing. CM has traditionally operated with source code elements. However, this is not sufficient anymore. All item types produced during the development need to be managed including requirements, management documents, designs, code, compilers, operating systems, binaries, instructions, manuals, etc. These items have different physical characteristics affecting how they should be identified, stored in the software library and used to define and build configurations (Whitgift 1991). As stated above in section 2.2, the different types of configuration items have different needs for control. Therefore, the categories and item types per each category need to be determined (Buckley 1996) to allow proper identification and control for each type of items.

3.1.10 Management Constraints on the CM Plan

According to Whitgift (1991), it is unusual that the author of a CM plan starts with a blank sheet of paper. The use of the existing CM templates as a starting point for the planning of the CM practices is stated in literature (e.g. Whitgift 1991, Leon 2000). Usually, the CM plan must also comply to the imposed or established working methods and tools. The working methods might even come from the customer or from the larger organisation of which the project is a part (Whitgift 1991). Also, consistency between the management plans is important and the CM plan should refer to the other plans such as the project plan and the quality plan rather than reiterate them. As stated above, *a successful CM plan reflects its project environment* (IEEE Std-828 1998). Therefore, the content of the CM plan should be synchronized with all changes in this environment during the life cycle of the project. The plan needs also to take into account the project resources, including the skills and background of the project team (Whitgift 1991).

3.2 Framework of Factors

This section collects the framework of factors based on section 3.1 (Table 1). These factors should be taken into account singly and together to allow the definition of the CM practices that will suit a product development project. The table also summarizes the initial set of guidelines for the CM solutions based on literature.

Table 1. Framework of factors for the adaptation of configuration management.

Factor	Description	Initial guidelines for CM	References
Size of the project	The size of the development project.	Small projects need less formal practices and a single person can be responsible for the CM coordination. Large projects need formal practices and a CM team.	Whitgift (1991), Leon (2000)
Product type	The type of the product that is being developed.	Need for formality and automation increases with the level of the product criticality.	Jonassen Hass (2003)
Project hierarchy (distribution)	Is the product development distributed over several interconnected projects?	Effective CM procedures are needed for each part of the distributed project. Interfaces between the sub-systems should be defined and controlled. CM plans indicating different levels of CM need to be compatible with each other.	Whitgift (1991), Abran and Moore (2004)
Multisite development	Is the development dispersed over several geographically distributed sites?	Address challenges of networking, communication, security, and concurrency management in the multisite development. The information needs to be shared and responsibilities made clear in the multisite environment. Need for formality and automation (tool support) increases when moving from local development environment to global. The same process should be followed in all sites if possible. Enforcement that CM is practiced according to the rules in all sites.	Leon (2000), Jonassen Hass (2003), Ebert and De Neve (2001), Battin et al. (2001), Rahikkala (2000)
Development disciplines	Are there several developmental disciplines (SW, electronics, and mechanics) that work together for a common system product?	Consistent identification practices and careful change management are needed for those items that are naturally related (HW/SW). Ensure good communication between HW and SW elements of CM during the CM planning.	Crnkovic et al. (2003), Jonassen Hass (2003), Lyon (1999)
Development models	Models, methods, approaches that have been selected for the product development.	The characteristics of the development models shape the CM solutions. E.g. an iterative development requires a good version management and baselining to ensure sufficient traceability.	Jonassen Hass (2003), Ronkainen and Abrahamsson (2003)

Factor	Description	Initial guidelines for CM	References
Dependence on third party software	Third party SW is used in SW development. Possible new versions.	Practices of how the third party software is to be specified, delivered, accepted and changed. Ensure that all development teams access the right versions of the components to avoid problems in integration.	Whitgift (1991), Berczuk and Appelton (2003)
Maintenance and multivariants	Several customers with customer-specific features. Maintenance of systems.	Define the practices how the variants are managed and releases of the system to different customers built and recorded. Define the practices how old versions of the systems, which are still in the operational use, are maintained and for how long.	Whitgift (1991), Jonassen Hass (2003)
Item types	Different kinds of item types are developed during the product development.	Manage all product-related items produced during the development. Identify item categories and item types per each category to allow proper identification and control for each type of items.	Whitgift (1991), Buckley (1996)
Management constraints on the CM plan	Working methods, other management related plans and the competence of the resources.	The CM plan needs to comply to the imposed or established working methods and tools. The CM plan should reflect its project environment. Take into account the project's resources, including the skills and background of the project team. The existing CM templates are a good starting point for the planning of CM practices.	Whitgift (1991), IEEE Std-828 (1998), Leon (2000)

3.3 Summary

The context of the development project sets various constraints for CM. Therefore, CM is a situation dependent activity that needs to face these factors to allow efficient practices for managing the system during its life cycle. This section constructed the initial framework of factors that affect the CM solution. This framework has been used for analysing CM practices in three SW development projects (see the section 4). Even though a set of these factors can be identified, the challenge is that they need to be taken into account singly and together to allow the definition of the CM practices that will suit a product development project.

4. Empirical Evaluation of Configuration Management Adaptation

This section considers three CM cases. First the cases are described using the framework defined in the previous section. Then the CM observations made of the cases are described and analysed. A cross-case analysis is used to examine the similarities and differences of the cases.

4.1 Case Study Characterisation

This section describes three case study projects according to the factors presented in section 3. The following Table 2 depicts the summary of the project characteristics. The cases are on columns and factors on rows. The sub-sections discuss each case in more detail.

Table 2. Description of cases.

Cases / Factors	Case 1:	Case 2:	Case 3:
Size of the project	Large DSP (Digital Signal Processing) SW project.	Small.	Small.
Product type	The project was a part of the radio base station development.	System for managing the research data and documents.	Financial sector mobile SW.
Project hierarchy (distribution)	Work is organised around hierarchical projects and teams.	One project.	One project. Customer was not present.
Multisite development	Several geographically distributed sites.	Local.	Local.
Different disciplines	SW development and connections to HW development.	SW development.	SW development.
Development models	Iterative development approach. HW/SW codesign.	Agile-based development.	Agile-based development.
Dependence on third party software	Third party components.	Tool versions remained the same.	Tool versions remained the same.
Maintenance and multivariants	Long product life duration.	Information retrieval was needed for reuse and research reasons.	Information retrieval was needed for reuse and research reasons.
Item types	Several item types.	Several item types.	Several item types.
Management constraints on the CM plan	Company policy for the CM plans.	No company policies related to the CM plans.	The CM plan adopted from a previous project (eXpert).

4.1.1 Case 1

Case 1 related to the adaptation of CM for a project developing DSP SW and HW related DSP SW. Case 1 is reported in detail in Paper I. The paper reports experiences related to the embedded systems' CM in a large project in concurrent development environment. Furthermore, the project was a part of the radio base station development.

The development context of case 1 was a complex. The development was a part of a larger project hierarchy where the development of the SW was divided into several system components (e.g. DSP SW, Protocol SW and Operation and Maintenance (O&M) SW). Furthermore, the development of DSP SW was divided into several geographically distributed sites. The DSP functionality solved by algorithms was divided into SW and HW. Operations with critical execution speed are usually allocated to HW. HW is realised as ASICs (Application Specific Integrated Circuits) or FPGAs (Field Programmable Gate Arrays), if there is a need for field programmability. Therefore, there were interconnections between HW and DSP SW development.

The overall development paradigm was HW/SW codesign. The nature of HW/SW codesign is experimental and iterative (Ronkainen and Abrahamsson 2003). There are several communication points during the development process where the SW and HW configurations are simulated against each other. SW development also comprises other types of elements and not just product files. Compilers, operating systems, etc. are also elements that may evolve during the product development. The radio base stations have a long life duration and therefore maintenance is an issue for these products. The configuration item types that were produced comprise different types of elements, such as text documents, diagrams, source code, etc. To assist the planning of CM, the company had existing templates and processes for CM including a pre-installed CM tool.

4.1.2 Case 2

Case 2 related to the adaptation of CM for a small (less than 10 persons) project (eXpert) that used agile development principles for producing a SW system for

managing the research data and documents in VTT. The case is reported in Paper II and Paper III.

The context of case 2 was much simpler than in case 1. The development was organised around one project working physically in one location and the customer's representative was also practically always present. The project developed SW following an agile-based development method (Extreme Programming (XP)) that brought about the challenges of iterative and fast-paced development. The development tool versions remained the same. Although the team did not have long maintenance responsibilities, the retrieval of the product information was needed for reuse and research purposes. The development process produced several types of items, such as management documents, product code and test code. However, requirement specification was practiced using a paper-pen board method and therefore specifications were not in the electronic format. The organisation did not have policies for CM, which meant that the planning of CM was started from scratch. Also, the CM tool had to be selected and installed for the project. Persons that were involved in the CM planning had the background of a plan-based, bureaucratic CM.

4.1.3 Case 3

Case 3 related to the adaptation of CM and improvement of RM automation in a project (zOmbie) that used agile development principles for developing a financial sector mobile SW. The CM lessons learned have been collected based on the interview of the CM expert of the project. The improvement attempt of the RM is reported in Paper IV. Cases 2 and 3 form a project continuum in VTT where several experimental SW development projects were established to examine the utility of the agile development methods.

The context of case 3 was in many respects similar as in case 2. Just a few factors were different. First, the project produced a different type of product than in case 2 (financial sector mobile SW). Second, the customer's representative was not present, but worked off-site. The most remarkable difference was that the CM plan template and practices produced during case 2 could be used for tailoring the CM practices for the zOmbie-project. Also, the CM tool and new electronic solution for the management of requirements were pre-installed for the project.

4.2 Results

This section presents results of the analysis when the lessons learned related to the configuration management in case studies are mapped with the factors. The results are described in tables as follows:

- The first column represents the lessons learned related to the CM practices in the projects.
- The second column classifies the CM practices according to the CM elements presented in section 2.
- The third column depicts an interpretation on which factors have affected the formation of the CM practices.

4.2.1 Case 1

The following Table 3 shows the results of the analysis of case 1. It depicts an interpretation of how certain project-dependent factors affect the applicability of the CM practices. Each of these CM practices will be discussed. The CM plan of case 1 is confidential information of the company, but the detailed CM experiences have been documented in Paper I.

Table 3. Mapping between the CM practices and project factors.

CM practices	**CM element reference**	**Project factor reference** ***(project characteristics are in brackets)***
Company/project level CM organisation	CM planning	– Size of the project *(large)* – Project hierarchy *(distributed over several projects)*
Cooperative CM planning	CM planning	– Project hierarchy *(distributed over several projects)* – Development disciplines *(SW, relation to ASIC development)*
Clear responsibilities for CM tasks	CM planning	– Size of the project *(large)* – Project hierarchy *(distributed over several projects)* – Development disciplines *(SW, relation to ASIC development)*
Managing change in a distributed environment	Configuration control	– Project hierarchy *(distributed over several projects)* – Development disciplines *(SW, relation to ASIC development)* – Dependence on third party software *(third party components, such as compilers, etc.)*
Information sharing	Configuration identification	– Project hierarchy *(distributed over several projects)* – Multisite development *(several sites)* – Development disciplines *(SW, relation to ASIC development)*
Traceability, baselining	Configuration identification	– Project hierarchy *(distributed over several projects)* – Development disciplines *(SW, relation to ASIC development)* – Development models *(HW/SW codesign)* – Maintenance and multivariants *(long product life duration)*
Managing iterative SW design	Configuration identification	– Development models *(HW/SW codesign)*
CM enforcement (system configuration audit)	Configuration audit, Configuration status accounting	– Project hierarchy *(distributed over several projects)* – Multisite development *(several sites)* – Management constraints on the CM plan *(company policy for CM plans)*
Make the items in product life cycle identifiable and manageable	Configuration identification, Configuration control	– Project hierarchy *(distributed over several projects)* – Development disciplines *(SW, relation to ASIC development)* – Dependence on third party software *(third party components, such as compilers, etc.)* – Item types *(several types in different categories)*
Plan-based CM	CM planning	– Management constraints on the CM plan *(company policy for CM plans)*

Company/project Level CM Organisation

The organisation produces complex products and projects tend to be large. The *organisation of CM was divided into the organisation- and project–level*. The projects have the best understanding of the development tools they are using and what kind of sub-systems they are producing. Therefore, the project level CM teams tailor CM for the projects according to the CM plan template and coordinate CM inside the projects. The organisation level team is needed to improve and generalize CM for the organisation (e.g. CM plan templates), provide CM consultation and technical support for the projects, as well as coordinate overall CM in the organisation.

Cooperative CM Planning

The project environment contained several inter-connected projects. This caused the need for *co-planning and communication between the projects to resolve the common CM issues* (e.g. management of interfaces (common configuration items)). This communication is already needed in the CM planning phase. One "CM forum" was used as an inter-project CM planning team to resolve and communicate the common CM planning issues as well as to share the experiences of the successful CM practices between the projects. It is important that the CM forum assemble for a meeting on a regular basis in the CM planning phase. On the other hand, afterwards, the CM forum could meet less frequently. In order to achieve compatible CM practices over the development disciplines, a good communication between the interconnected HW and SW teams is crucial.

Clear Responsibilities for the CM Tasks

Clear responsibilities for the CM tasks are important in a project environment where the systems under development are large with several sub-systems and implementation technologies. *CM inside a project is quite easy, but the complexity increases when the development is organised into separate hierarchical development projects and teams.* Without clear responsibilities for the CM activities in this multiproject environment, CM might be well realized inside a project, but the common CM issues (management of interfaces (common configuration items, interface specifications, etc.)) are insufficient leading to problems in the integration and maintenance phases.

Managing Change in a Distributed Environment

The sub-systems produced by projects and teams have dependencies and *changes in one of the sub-systems can cause changes in the other*. For example, changes to the SW may cause changes to the HW and vice versa. Thus, change management practices and responsibilities should be defined in such a way that ensures controlled change documentation, evaluation, implementation and notification through the related projects. In particular, *the impacts to other projects should be analyzed to avoid inconsistency during the integration*. Items that require special attention here are *SW/SW interface items, HW/SW interface specifications/items and third party components* to avoid uncontrolled changes in these items.

Information Sharing

Capabilities for *information sharing and management (e.g. using replication) are important in a project environment where the development is dispersed* into several projects and disciplines as well as over several geographically distributed sites. Proper level of information visibility is crucial to prevent the isolated working culture where changes are not communicated to all relevant parties. Furthermore, there need to be practices how the interfaces are defined and controlled when different sub-systems are developed on different sites.

Traceability, Baselining

Several interconnected SW development projects make it necessary that the *SW-SW interface items are given special attention*. It is vital that these items have been identified and all participants have the correct versions (baseline of common items) of the common items to ensure the consistency of the product during the integration phase. Also the *traceability of the product information* is essential for maintenance purposes when operating with products that have long product life duration. Furthermore, cooperative HW/SW development where SW and HW configurations are simulated iteratively requires attention. *The releases should be based on baselines* and there should be clear practices on how to identify and document releases to ensure their sufficient traceability. *The compatibility of SW and HW sub-releases during simulation needs to be documented for traceability*.

Managing Iterative SW Design

Incremental and cooperative HW/SW development creates concurrency to the SW development. This might make it necessary to have a *parallel bug-fixing branch* for a bug-fix, when development and simulation happen concurrently. The bug-fixing of the previous release might happen in the bug-fixing branch, which will be *merged into the main branch* before the next release to ensure that the problems are fixed in the upcoming release.

CM Enforcement (System Configuration Audit)

Mechanisms for monitoring that consistent CM practices are followed are needed in the complex development environment where the development is divided over interconnected projects and distributed over several sites. This is needed, for example, to monitor that all SW elements are placed under the configuration management. Furthermore, during long projects, *CM should be able to response to any changes in the project's operational environment* that might affect the CM practices or tool support.

Make the Items in Product Life Cycle Identifiable and Manageable

The role of configuration management is to provide mechanisms to *identify, store and manage configuration items during product development* from the requirements specifications to the implementation. Furthermore, the proper identification and management of items requiring special attention is important. These kinds of items are, for example, the *SW/SW and HW/SW interface items* as well as *third party components*, such as tools and operating systems. Without proper identification and coordination some teams might start to use, for instance, the newer version of a common component without notifying the other parties about the changes, which might lead to problems when integrating the sub-systems.

Plan-based CM

The use of *templates and processes for planning CM for the projects makes the whole adaptation easier*. Templates provide "tried and tested" general frameworks for defining the CM practices that suit the special characteristics of the projects. In this case the CM activities planned for the project covered the traditional CM issues

described in section 2 (organisation, identification, control, status accounting, etc.). It became evident that nowadays inside a project, CM is usually fairly well realised, but the complexity and challenges of CM come from the size of the project (large), project hierarchy, multisite development and development disciplines. Furthermore, the *pre-installed full-fledged CM tool* was an essential element to support configuration management in the complex project environment.

4.2.2 Case 2

The following Table 4 shows the results of the analysis of case 2. It depicts an interpretation of how certain project-dependent factors affect the applicability of the CM practices. Each of these CM practices will be discussed. Detailed CM experiences have been documented in Paper II and the RM procedures used have been documented in Paper III.

Table 4. Mapping between the CM practices and project factors.

CM practices	**CM element reference**	**Project factor reference** ***(project characteristics are in brackets)***
Manual requirements management	Configuration identification	– Project hierarchy *(one project)* – Multisite development *(local)* – Development models *(agile)*
Lightweight change management	Configuration control	– Size of the project *(small)* – Development models *(agile)*
Flexible CM tool	CM planning	– Size of the project *(small)* – Development models *(agile)* – Management constraints on the CM plan *(no company policies for CM)*
Parallel development branches (simultaneous modifications)	Configuration identification	– Development models *(agile)*
Traceability, baselining	Configuration identification	– Development models *(agile)* – Maintenance and multivariants *(information retrieval for reuse & research purposes)*
Making the items in product life cycle identifiable and manageable	Configuration identification, Configuration control	– Item types *(several item types)*
CM enforcement (system configuration audit)	Configuration audit, Configuration status accounting	– Management constraints on the CM plan *(no company policies for CM)*
Plan-based CM	CM planning	– Management constraints on the CM plan *(no company policies for CM)*

Manual Requirements Management

The functional baseline was managed manually using paper cards (paper-pen board method) as described in the XP method's practices. In many respects, the manual practice was sufficient. This kind of a solution is possible when a small project operates physically in the same location. However, this practice was identified as an improvement area to achieve more control for the requirements. This is especially needed when some stakeholders (e.g. customer's representative) are physically in a different location from the development team.

Lightweight Change Management

The development team was quite small and the development method stressed *lightweight practices and documentation*. Therefore, special attention was paid to "simple" change management practices.

Flexible CM Tool

A flexible, easy to use CM tool is essential in a fast agile-based development. The selected tool was CVS, which is not a full-fledged CM tool, but rather a version management tool. The project team used the Eclipse tool integration framework as a development environment in the project. *CVS was pre-installed into the Eclipse environment* which affected the tool selection most. The selected tool is *suitable especially for a small team*.

Parallel Development Branches (Simultaneous Modifications)

Support for *simultaneous updates was found essential* in order to support iterative development and XP practices, such as continuous integration and collective ownership. This created special requirements for the CM tool to allow for the *optimistic version control mechanisms and good merge support* if conflicts were to occur in the check-in operation.

Traceability, Baselining

Generally, only the last versions of the files in the repository were relevant, but *some exceptions required that the return to the old versions was needed*.

Baseline was created in every iteration. The project history revealed that there were some problems in making corrections (bug-fixing) to the product after the last release, because the final product baseline was not done.

Making the Items in Product Life Cycle Identifiable and Manageable

Furthermore, effective CM requires that *all types of developmental artefacts be placed under the configuration management*. However, especially for some management documents this was not always done, which caused problems.

CM Enforcement (System Configuration Audit)

In agile development, a team reflects on how to become more effective and then adjusts its behaviour accordingly. There is also a need to monitor that everyone is operating according to the rules; for example, that all SW elements are placed under configuration management and coding standards were followed. To support these issues, *the team used system configuration audits and made corresponding corrective actions and modifications to the CM plan when needed*. Generally the use of configuration audits had a positive influence to the quality of SW releases in eXpert-project.

Plan-based CM

The organisation did not have any policies for CM and, therefore, the development needed to start from scratch. *IEEE Std-828 -template (1998) was the basis for creating a company-specific CM template*. The template served as an excellent basis for the CM planning, showing the CM practices that need attention. The irrelevant parts of the template, such as subcontractor/vendor control and interface control were ignored. This template was then tailored for a project including responsibilities, schedules and project specific CM practices.

4.2.3 Case 3

This section discusses the results of the analysis of case 3 (Table 5). Case 2 and 3 were quite similar with almost the same project characteristics. Therefore, this

section only analyses those CM practices that differ from the CM practices in case 2. The only exceptions were that:

- The CM plan was based on the plan used in case 2 with minor modifications. Also, the RM tool was constructed for the project. These caused deviations to the factor "Management constraints on the CM plan" if compared to case 2.
- The customer's representative was not always present in the developer's workplace as he was in case 2. This caused deviation to the factor "Project hierarchy (distribution)", if compared to case 2.
- The "product type" was different from case 2.

Table 5. Mapping between the CM practices and project factors.

CM practices	**CM element reference**	**Project factor referenc** ***(project characteristics are in brackets)***
Electronic requirements management	Configuration identification	– Project hierarchy *(one project, customer was not present)* – Multisite development *(local)* – Development models *(agile)* – Management constraints on the CM plan *(CM plan copied from earlier project)*
Simple change management	Configuration control	– Size of the project *(small)* – Development models *(agile)*
Easy to use CM tool	CM planning	– Size of the project *(small)* – Development models *(agile)* – Management constraints on the CM plan *(CM plan copied from earlier project)*
Parallel development branches (simultaneous modifications)	Configuration identification	– Development models *(agile)*
Traceability, baselining	Configuration identification	– Development models *(agile)* – Maintenance and multivariants *(retrieval for reuse & research purposes)*
Making the items in product life cycle identifiable and manageable	Configuration identification, Configuration control	– Item types *(several item types)*
CM enforcement (system configuration audit)	Configuration audit, Configuration status accounting	– Management constraints on the CM plan *(CM plan copied from earlier project)*
Plan-based CM	CM planning	– Management constraints on the CM plan *(CM plan copied from earlier project)*

Electronic Requirements Management

Case 3 contained the improvement attempt of RM (experiences from the improvement attempt are documented in Paper IV). This was needed because the customer's representative of the project was not present in the same location as the development team. This meant that a need emerged for more effective communication and data visibility mechanisms than in case 2 (management of requirements data). However, the development team worked in one office room, and therefore, the challenge was to provide a tool that was as easy and flexible to use as the manual solution. A tool was constructed for the more automated, i.e. electronic, management of requirements (described in Paper III). The tool provided a possibility for describing, organising and baselining requirements. The tool was integrated into the Eclipse environment enabling the project team to work with one channel throughout the whole development life cycle and providing access to the requirements information over the network for the customer's representative. However, the tool was not mature enough. It was found too difficult to use and it failed to provide as powerful a visual view as the manual paper-pen board method for the local agile development team. Therefore, the project team returned back to the manual RM practices. The tool, constructed to support any development method, should take into account the underlying values of the method itself or the tool fights against the nature of the method. This deficiency has been resolved in future projects with a new approach to RM combining the manual and electronic solutions to enable the external stakeholders easily to view the status of the requirements implementation.

Plan-based CM

The CM plan was based on the plan used in case 2 with minor modifications. Therefore, the CM planning phase was quite straightforward and the pre-installed CM tool could be used. In the CM system audits, there were mainly some minor modifications to the project's CM practices, for example, some modifications to audit templates and procedures. The only remarkable change was the project team's decision to return back to the manual RM practices when the RM tool turned out to be unsuitable for the project. However, these CM practice modifications were not documented to the CM plan anymore, because the update procedures were found too strict. Furthermore, the interview of the

project's SCM expert revealed that in this case, when CM was turned out to be quite stable, the documented CM plan was not an issue but "standardised" training and tool support were the elements that replaced the extensive CM documentation. In other words, the CM practices were somewhat stabilised as a part of the CVS use and a separate CM plan was not updated anymore. The successful CM practices were introduced as a part of the description of the development method in future projects.

4.2.4 Cross-case Analysis

Cross-case analysis is used to analyse the similarities and differences between the cases. The following Table 6 summarises the results of the analysis. Cases 2 and 3 represent a project continuum in the same organisation and therefore they do not have many differences.

Table 6. CM observations.

Cases / Factor	CM observations from case 1:	CM observations from case 2:	CM observations from case 3:
Size of the project	Large project => *formality and clear responsibilities for the CM activities are needed.*	Small project => *less formal practices, a simple CM tool.*	Small project => *less formal practices, a simple CM tool.*
Product type	SW was a part of a radio base station development => c*ase study did not provide clear evidence on how the product type affects the CM practices.*	Document management system => c*ase study did not provide clear evidence on how the product type affects the CM practices.*	Financial sector mobile SW => c*ase study did not provide clear evidence on how the product type affects the CM practices.*
Project hierarchy (distribution)	Distributed over several projects and teams => *In distributed projects the interconnections require* • *clear responsibilities and enforcement for the CM activities* • *identification and management of the interfaces (e.g. interface specifications and common configuration items)* • *controlled change documentation, evaluation, implementation and notification through related projects* • *cooperation in the CM planning to ensure consistent practices* • *information visibility for the relevant parties*	One project => *not an issue.*	One project, customer was not present => *need for information visibility for the relevant parties.*
Multisite development	Several sites => *In a multisite environment, proper information sharing, interface management and system configuration audit mechanisms are needed.*	One site => *not an issue.*	One site => *not an issue.*
Different disciplines	SW development and connections to HW development => *Interconnections requireclear responsibilities and cooperation for the CM activities* • *identification and change management for interfacing items,* • *traceability of the related HW/SW packages* • *cooperation already in the CM planning phase*	SW development => *not an issue.*	SW development => *not an issue.*

Cases / Factor	CM observations from case 1:	CM observations from case 2:	CM observations from case 3:
Development models	Iterative development approach with HW/SW codesign => *Iterative development approach with HW/SW codesign requires support for the parallel SW development (branching and merging) and clear baselining for traceability.*	Agile-based development method => *Agile-based development method requires support for the simultaneous updates, traceability, baselining and simple easy-to-use practices and tools.*	Agile-based development method => *Agile-based development method requires support for the simultaneous updates, traceability, baselining and simple easy to use practices and tools.*
Dependence on third party software	Third party components, such as, compilers and operating systems => *Third party components require strict identification and management of the versions.*	Tool versions remained the same => *not an issue.*	Tool versions remained the same => *not an issue.*
Maintenance and variation	Long product life duration => *abilities for information retrieval.*	*For reuse and research reasons the information needs to be retrieved.*	*For reuse and research reasons the information needs to be retrieved.*
Item types	Several item types => • *Item categories and types need to be identified so that the configuration items can be managed and retrieved effectively during the life cycle of the project.* • *Put special attention to common configuration items, interface specifications and third party SW.*	Several item types => *Item categories and types need to be identified so that the configuration items can be managed and retrieved effectively during the life cycle of the project.*	Several item types => *Item categories and types need to be identified so that the configuration items can be managed and retrieved effectively during the life cycle of the project.*
Management constraints on the CM plan	Company policy for the CM plans, a pre-installed CM tool => • *Company policies for the CM plans and a pre-installed CM tool are essential elements for complex organisations. These elements speed-up the CM planning.* • *Changes in the project environment need to be reflected in the CM practices and tool support (system CM audits).*	No company policies related to the CM plans => • *If the company does not have policies related to the CM plans it is good to start the CM planning based on a standard template.* • *Changes in the project environment need to be reflected in the CM practices and tool support (system CM audits).*	CM and development environment adopted from case 2 => *Reuse of the successful CM practices from previous projects.*

Some similarities can be found between cases even though the context of case 1 was in many respects different from that of cases 2 and 3.

In all cases, the plan based CM worked well. Basic CM activities, for example, configuration identification (naming, versioning, item acquisition) and configuration control (change management) existed in all cases, but the practical procedures, formality and need for automation varied depending on the project environment.

Cases 2 and 3 represented a project continuum in the same organisation (VTT). This enabled the improvement of the CM practices from project to project using stepwise approach. The CM plan based approach was essential in case 2 when the organisation did not have any pre-defined policies and tools for CM. On the other hand, in case 3, formal CM plan document updates became unnecessary. Later agile-projects in VTT, the CM practices were embedded into the development method, and a separate CM plan -document does not exist anymore. This could have been possible because of the project size (small) and method (agile) used.

Clear evidence on how the product type has affected the CM solution was not found in this study. Jonassen Hass (2003) mentions that it is difficult to provide unambiguous rules for CM for different classes of products. But it can be assumed that in case 1 the malfunction in a product can cause a major financial loss for a customer and therefore the need for the formality of CM increases.

Furthermore, case 1 represented a project that was a part of a larger development project. This caused the need for co-planning and communication between projects to resolve the common CM issues (e.g. management of interfaces). This kind of co-planning challenge was not an issue in cases 2 and 3.

Only case 1 was a truly multisite project that needed practices and technical solutions for multisite development (e.g. replication, branching). Case 2 was totally local, but case 3 had the off-site customer. This meant that a need emerged for more effective communication and visibility of the requirements in case 3 than in case 2 (management of requirements data). An electronic computer-based solution was built to resolve the problem. However, the solution was not sufficiently simple for the agile-type fast-paced development. The results emphasize the role of adaptation in the tool development. The tool, constructed to support any development method, should take into account the underlying values of the method itself. Without this, the tool fights against the nature of the method. This deficiency was resolved in later projects with a new approach to RM which combined the manual and electronic solution.

Different development disciplines are challenging in CM. Different kind of terminology and tools are used for the development and, in many cases, HW and SW teams are physically in different locations, which prevents natural

interactions. In cases 2 and 3, this was not an issue as it was in case 1, where the need for the management of interfaces and support for information exchange were observed.

Case 1 and cases 2 and 3 used different SW development approaches. At first glance, these approaches seem to be totally different, but actually they have some characteristics in common that led to similar CM solutions. The development of embedded SW (especially HW/SW codesign) is experimental and iterative, requiring an efficient tool support for version management and baselining. This is also true for the agile development methods, especially for XP. The importance of a fluent CM tool support was emphasised in all cases.

Third party components, such as compilers, operating systems, etc. should be under CM if new versions emerge. This was highlighted in case 1, while it was not an issue in cases 2 and 3. The need for information retrieval for maintenance reasons was obvious in case 1. For cases 2 and 3, the information retrieval is topical mainly for reuse and research reasons. However, in case 2, there were some problems in making corrections (bug-fixing) to the product after the last release, because the final product baseline was not done. This is a typical occasion when the importance of traceability and baselining is emphasised. Furthermore, configuration management has traditionally mainly operated with the source code elements. However, this is not sufficient anymore. All item types produced during the development need to be managed including requirements, management documents, designs, code, compilers, operating systems, binaries, instructions, manuals, etc. The importance of the management of all developmental artefacts was detected in this analysis.

5. Introduction to the Papers

This chapter introduces the papers that are included into this research. The papers relate to the cases described in this study as follows:

- Paper I relates to case 1
- Papers II and III relate to case 2
- Papers III and IV relate to case 3.

The contexts of the papers are illustrated in the following Figure 6:

	"Traditional" SW development	Agile SW development
Tool support	Paper I	Papers II, III, IV
Procedures	Paper I	Papers II, III

Figure 6. Contexts of the papers.

5.1 Paper I: *Configuration Management Support for the Development of an Embedded system: Experiences in the Telecommunication Industry*

Paper I presents the importance and practical experiences of configuration management in the context of HW related SW development (DSP SW and HW related DSP SW). First, the paper provides an introduction, based on a literature study, to the concepts of CM and HW related SW development, and then discusses the operational environment of CM in an organisation developing telecommunication products. The paper highlights the complexity of the

development of embedded systems. The paper describes the observations that were made during the CM planning, setting, implementation and modification. Requirements management is limited to cover the management of specification documents. The work provides a case for the analysis of the CM adaptation in a complex project environment.

5.2 Paper II: *Improving Software Configuration Management for Extreme Programming: a Controlled Case Study*

Paper II presents a case study where the CM principles are applied for a software project (called "eXpert") that adhered to the agile-based development method. The project produced a system for managing the research data and documents. The study started with a literature study following a case study where the CM principles were planned for a SW project. The paper presents how CM was planned and realised to support the management of the product information in the project. The work provides a case (case 2) for the analysis of the CM adaptation in agile environment.

5.3 Paper III: *Supporting Requirements Engineering in Extreme Programming: Managing User Stories*

Paper III presents a proposed tool for managing the requirements in the context of the XP method. The paper first presents the basic concepts of RM and introduces the traditional manual RM procedures that were used in VTT in the first XP-based SW development project (eXpert). The paper then analyses the comments from the project and lay out the requirements for RM tool support in the XP development environment. The paper further introduces a proposed tool for managing the requirements in the XP project. The work is a part of a case (case 2) described in paper II extending it with functional baselining issue that is an essential part of CM. The work also provides a starting point and describes a proposed RM tool for case 3.

5.4 Paper IV: *Improving Requirements Management in Extreme Programming with Tool Support – an Improvement Attempt that Failed*

Paper IV presents experiences while using a tool presented in paper III in a project developing mobile application software (financial sector mobile SW) using an agile based development method (zOmbie -project). The tool turned out to be too complicated and did not provide enough support for the fast-paced development. The paper discusses issues that led to this dissatisfaction. The paper further identifies issues that should be taken into account when considering RM for a fast-paced development environment. The paper also highlights that it is important to understand the development method used in a project to avoid the RM or CM solutions that work against the underlying nature of the development method. The RM improvement work was done using action research -method. This work with the interview of zOmbie-project's CM expert provides another case (case 3) for the analysis of the CM adaptation in agile environment.

6. Conclusions and Future Research Needs

This research examined the factors that affect the practical configuration management solutions. Set of factors were presented as a framework and they were used to analyse three case studies to better understand how the factors affect the CM solutions. The study provides guidelines for the CM solutions that are based on the experiences of real life CM cases. The following sections first discuss the results and their implications. Then, the research questions are answered. Finally, future research needs are indicated.

6.1 Evaluation of Results

This section discusses the results of the study and the generalisation of the results. Table 7 depicts the initial and case-based guidelines for the CM defined in this study.

Table 7. Initial and case-based guidelines for CM.

Factor	Initial guidelines for CM	CM guidelines based on three case studies
Size of the project	Small projects need less formal practices and a single person can be responsible for the CM coordination. Large projects need formal practices and a CM team.	In a small project, less formal practices and a simple CM tool are adequate. In a large project, formality and clear responsibilities for the CM activities are needed.
Product type	Need for formality and automation increases with the level of the product criticality.	Case studies did not provide clear evidence on how the product type affects the CM practices.
Project hierarchy (distribution)	Effective CM procedures are needed for each part of the distributed project. Interfaces between the sub-systems should be defined and controlled. CM plans indicating different levels of CM need to be compatible with each other.	In distributed projects the interconnections require • clear responsibilities and enforcement for CM activities • identification and management of interfaces (e.g. common configuration items) • controlled change documentation, evaluation, implementation and notification through related projects • cooperation in CM planning to ensure consistent practices • information visibility for relevant parties
Multisite development	Address challenges of networking, communication, security, and concurrency management in multisite development. The information needs to be shared and responsibilities made clear in a multisite environment. Need for formality and automation (tool support) increases when moving from a local development environment to a global one. The same process should be followed in all sites if possible. Enforcement that CM is practiced according to the rules in all sites.	In a multisite environment, proper information sharing, interface management and system configuration audit mechanisms are needed.
Different disciplines	Consistent identification practices and careful change management are needed for those items that are naturally related (HW, SW). Ensure good communication between HW and SW elements of CM during the CM planning.	Interconnections require • clear responsibilities and cooperation for the CM activities • identification and change management for interfacing items • traceability of related HW/SW packages • cooperation already in CM planning phase

Factor	Initial guidelines for CM	CM guidelines based on three case studies
Development models	The characteristics of the development models shape the CM solutions. E.g. an iterative development requires a good version management and baselining to ensure sufficient traceability.	The iterative development approach with HW/SW codesign requires support for parallel SW development (branching and merging) and clear baselining for traceability. The agile-based development method requires support for the simultaneous updates, traceability, baselining and simple easy-to-use practices and tools.
Dependence on third party software	Practices how third party software is to be specified, delivered, accepted and changed. Ensure that all development teams access the correct versions of the components to avoid problems in the integration.	Third party components, such as compilers and operating systems, require strict identification and management of the versions.
Maintenance and variation	Define practices how variants are managed and releases of the system to different customers built and recorded. Define practices how old versions of the systems, which are still in operational use, are maintained and for how long.	Long product life duration requires abilities for information retrieval (traceability) for maintenance reasons. Information retrieval might also be needed for reuse and research reasons. The CM evidence for the variation management was not found (customisation was not an issue in the cases).
Item types	Manage all the product-related items produced during the development. Identify item categories and item types per each category to allow proper identification and control for each type of items.	The item categories and types need to be identified so that the configuration items can be managed and retrieved effectively during the life cycle of the project. Pay special attention to common configuration items, interface specifications and third party SW.
Management constraints on the CM plan	The CM plan needs to comply to the imposed or established working methods and tools. The CM plan should reflect its project environment. Take into account the project's resources, including the skills and background of the project team. The existing CM templates are a good starting point for the planning of the CM practices.	Company policies for the CM plans and a pre-installed CM tool are essential elements for complex organisations. These elements speed-up the CM planning. If the company does not have policies related to the CM plans, it is good to start the CM planning based on a standard template. Changes in the project environment need to be reflected on the CM practices and tool support (system CM audits). Reuse of successful CM practices from previous projects.

The CM guidelines gained from the three case studies are in many respects the same as the initial CM guidelines collected in section 3. Case-based CM

guidelines that were found in this study are not in conflict with the initial CM guidelines. However, clear CM guidelines for certain factors could not be found. These factors were *product type* and *variation*.

CM observations from case studies did not provide clear evidence on how the product type affects the CM practices. Literature states that the need for the formality and automation increase with the level of criticality (Jonassen Hass 2003). Likewise, the effects of the product variation for CM practices could not be studied in this research, because the customisation was not an issue in the cases. However, it is clear that variants need to be managed and maintained as described e.g. in Whitgift (1991).

Basic CM activities that should be planned for projects are the same (e.g. defined in standard template IEEE Std-828), but the practical procedures, formality and needs for automation vary depending on the project environment. This has also been stated in literature, e.g. in Leon (2000). The study also revealed that inside a project, CM is usually fairly well realised, but the complexity and challenges of CM come from the size of the project (large), work distribution (project hierarchy, multisite development, dependence on third party software components) and development disciplines (HW/SW). Especially the management of interfaces was found crucial in this kind of environment. Without strict practices unmanaged interfaces can cause difficult problems in the integration phase. According to Estublier et al. (2002) the basic concepts of (software) configuration management have been settled and the new challenges of CM come from its relationship to other domains, such as product data management (connection to system and HW development), component-based software development and software architecture. Nowadays, tough competition requires more flexibility and abilities to adapt to new development approaches to survive in the global competition. This changing development environment demands that CM face these new requirements whenever they emerge.

The research has been limited to the initial set of factors as described in section 3. Other potential factors, such as the maturity of an organisation and working culture have not been considered. Therefore, there is still much work to be done in future research to collect the more complete set of factors and their corresponding CM guidelines.

Can we generalise the results of the study? Separate factors, such as multisite development, dependence on third party SW and development disciplines clearly indicate the need for special CM concerns. However, projects tend to be different with the special characteristics of the development environment. Therefore these "factor-specific" solutions can only be used as guidelines for CM and their applicability need to be considered in the context of a project. This is because detailed CM solutions are the result of the interaction of several factors and each factor coexists with other factors and they need to be taken into consideration singly and together (Jonassen Hass 2003). Also, the nature of the case study method limits the generalisation of results. According to Yin (1994) a case study *investigates a contemporary phenomenon within its real-life context*. The results of this study are the most valuable when seeking configuration management solutions for projects operating within a similar context as those presented in this research. Therefore, this study attempts to provide a practical viewpoint to the CM adaptation so that *CM practitioners can compare the operational environment of their projects with our case studies and apply the findings of this study to their daily work when appropriate.*

6.2 Answers to the Research Questions

Answers to the research questions that were laid in section 1 are the following:

What are the basic mechanisms for adapting configuration management?

The basic mechanism is the CM planning. The CM responsibilities and procedures are described in the CM Plan -document. The plan itself can be a separate document or be included into, for example, project plan or quality plan. The CM planning should be done for each project, even if CM is practiced as a company policy, to ensure that the overall judgements are made about the applicability of the policies. A generic CM template usually referenced in literature is IEEE Std-828 (1998). It has become a widely accepted basis for the planning of initial CM practices.

What kind of factors influence the configuration management solutions?

The framework of the initial factors that present the project characteristics that affect the CM solutions has been constructed in this research. This framework is presented in section 3.

How do these factors affect the CM planning and practical CM solutions?

The following empirical guidelines are collected based on the three case studies presented in this research:

CM planning:

- *Project hierarchy (distribution):* the development of complex systems is usually divided into several interconnected projects. The cooperation between the projects already during the CM planning phase is essential to form consistent CM practices between the related projects.
- *Different disciplines:* several interconnected development disciplines require cooperative CM planning. HW/SW codesign especially requires the defining of the connection points between the HW and SW development to allow consistent management of the HW/SW product information.
- *Management constraints on the CM plan:* if an organisation does not have existing policies for CM, the formal plan-based definition of the CM responsibilities and practices is particularly useful. This is true even if the operational environment of the project is simple with, for instance, a small local development team. The existing CM templates provide a pre-defined base for the initial planning activity.

CM activities (practical CM solutions):

- *Size of the project*: the formality of the CM solutions varies. In large projects there is a need for formal procedures and clear responsibilities for the CM planning and activities, whereas in a small project the procedures can be less formal.
- *Product type:* clear evidence on how the product type has affected the CM solution was not found in this study. However, according to literature, the

need for formality and automation increases with the level of criticality (Jonassen Hass 2003).

- *Multisite development:* information sharing, interface management and enforcement to operate according to the agreed rules (i.e. system configuration audits).

- *Project hierarchy (distribution), different disciplines and dependence on third party software:* Interconnections between projects, HW/SW codesign and third party components require that the interfaces (specifications and other interfacing items) are given special attention (proper identification, baselining and change management). Communication and information sharing mechanisms are needed to support working in a multidiscipline and multiproject environment. Furthermore, complexity requires that there are clear responsibilities, formality and enforcement (i.e. system configuration audit) of the CM practices.
- *Development models:* CM is a support activity for the product development. Therefore, the nature and special needs of the development method used in a project should be taken into account. For example, for agile-based development, the solutions should be flexible and easy to use and they should comprise efficient practices for simultaneous updates, traceability and baselining.
- *Maintenance and multivariants:* the archiving and retrieval of information is needed for maintenance, reuse and other reasons (e.g. research purposes in cases 2 and 3). The case projects did not customise the products for customers and, therefore, the evidence on how different product variants were managed was not found in this study.
- *Item types:* the item types being managed should cover all artefacts on development life cycle from requirements to maintenance. These should also include the interface items, tools and management documents.
- *Management constraints on the CM plan:* changes in the operational environment of the project require that there are mechanisms to adapt the CM practices and tool support on the fly.

6.3 Future Research Needs

The study generated several ideas for future research topics. First, the framework needs to be elaborated (new factors) and further work needs to be done to empirically validate how the factors affect the CM solutions. Second, the life cycle approach to product information management is interesting. The consistent management of information from initial product ideas to physical electronics products involves several types of information management domains. These domains comprise, among others, requirements management (RM), software configuration management (SCM), product data management (PDM), supply chain management, enterprise resource planning (ERP), and document management (DM). The term "product lifecycle management" (PLM) provides a generic frame of reference for the systems and methods that are needed for managing all product related data during the product's life cycle. The planning of PLM for an organisation is extremely complex with several interconnected organisational functions, such as development, management, marketing, sales, manufacturing, etc. Understanding the complexity of PLM and mechanisms for planning and discovering effective solutions for it are essential in order to form consistent support for the product information management of complex products. Third, interface management was found complex topic that needs more consideration. Fourth, configuration management has recently been studied in the context of agile methods. Practical CM solutions for agile projects have stressed simplicity and automation to support the fast-paced development. However, it would be interesting to study how the agile CM solutions can contribute to the CM of HW/SW codesign, since there are some similarities in agile-based development and HW/SW codesign.

References

Abrahamsson, P., Salo, O., Ronkainen, J. & Warsta, J. (2002) Agile Software Development Methods: Review and Analysis. Technical Research Centre of Finland, Espoo. VTT Publications 478. 107 p. ISBN 951-38-6009-4; 951-38-6010-8. http://virtual.vtt.fi/inf/pdf/publications/2002/P478.pdf.

Abran, A. & Moore, J. (Executive editors) (2004) SweBok: Guide to the Software Engineering Body of Knowledge. 2004 version, IEEE Computer Society.

Battin, R.D., Crocker, R., Kreidler, J. & Subramanian, K. (2001) Leveraging resources in global software development. IEEE Software, Volume 18, Issue 2, March – April 2001. Pp. 70–77.

Berlack, H. (1992) Software configuration management. John Wiley & Sons.

Bersoff, E., Henderson, V. & Siegel, S. (1980). Software Configuration Management – An Investment in Product Integrity. Prentice Hall.

Bersoff, E., Henderson, V. & Siegel, S. (1979) Principles of software configuration management. Prentice-Hall, Englewood Cliffs, New Jersey.

Berczuk, S. & Appelton, B. (2003) Software configuration management patterns: effective teamwork, practical integration. Boston, Addison-Wesley. 218 p.

Buckley, F. (1996) Implementing configuration management: hardware, software, and firmware. IEEE Computer Society Press, Los Alamitos.

Christensen, H.B. (2001) Tracking Change in Rapid and eXtreme Development: A Challenge to SCM-tools? Tenth International Workshop on Software Configuration Management.

Crnkovic, I., Asklund, U. & Persson Dahlqvist, A. (2003) Implementing and Integrating Product Data Management and Software Configuration Management. Artech House, Boston.

Crnkovic, I., Dahlqvist, A.P. & Svensson, D. (2001) Complex systems development requirements – PDM and SCM integration. In: Asia-Pacific Conference on Quality Software.

Dart, S. (1996) Best practice for a configuration management solution. In: Sommerville, I. (ed.) Software Configuration management. Lecture Notes in Computer Science, Vol. 1167. International workshop on software configuration management (SCM6), Berlin, Germany. Springer-Verlag, Heidelberg, Germany. Pp. 239–255.

Ebert, C. & De Neve, P. (2001) Surviving global software development. IEEE Software, Volume 18, Issue 2, March – April 2001. Pp. 62–69.

Estublier, J. (2000) Software Configuration Management: A Roadmap. Finkelstein, A. (ed.) The Future of Software Engineering, 22nd International Conference on Software Engineering (ICSE 2000).

Estublier, J., Leblang, D., van der Hoek, A., Conradi, R., Clemm, G., Tichy, W. & Wiborg-Weber, D. (2005) Impact of software engineering research on the practice of software configuration management. ACM Transactions on Software Engineering and Methodology (TOSEM). Volume 14, Issue 4, October 2005, ACM Press, New York, USA. Pp. 383–430.

Estublier, J., Leblang, D., Clemm, G.R., Conradi, R., van der Hoek, A., Tichy, W. & Wiborg-Weber, D. (2002) Impact of the research community for the field of software configuration management. ICSE 2002. Proceedings of the 24rd International Conference on Software Engineering. Pp. 643–644.

Hult, M. & Lennung, S. (1980) Towards a definition of action research: A note and bibliography. Journal of Management Studies, May 1980. P. 241–250.

IEEE Std-610.12 (1990) IEEE standard glossary of software engineering terminology.

IEEE Std-828 (1998) IEEE Standard For Software Configuration Management Plans.

IEEE Std-1042 (1987) IEEE guide to software configuration management.

ISO/IEC 10007 (1995) Quality management – Guidelines for configuration management, International Standard.

ISO/IEC 12207 (1995) Information technology – Software life cycle processes, International Standard.

ITEA, ITEA 2 brief: Investing in Software-intensive Systems – Investing in Europe's Future. ITEA (Information Technology for European Advancement). http://www.itea-office.org/documents/Publications/ITEA-2_brief.pdf, available 29.11.2005.

Jonassen Hass, A. (2003) Configuration Management Principles and Practice. Addison Wesley. 370 p.

Leon, A. (2000) A Guide to software configuration management. Artech House, Boston.

Lyon, D. (1999) Practical CM – Best Configuration Management Practices for the 21st Century. 2nd edition, RAVEN Publishing Company.

Mens, T. 2002 A state-of-the-art survey on software merging. IEEE Transactions on Software Engineering. Volume 28, Issue 5, May 2002. Pp. 449–462.

Moreira, M. (2004) Software configuration management implementation roadmap. John Wiley & Sons. 244 p.

Mäkäräinen, M. (2000) Software change management processes in the development of embedded software. Technical Research Centre of Finland, Espoo. 185 p. + app. 56 p. VTT Publications 416. 185 p. + app. 56 p. ISBN 951-38-5573-2; 951-38-5574-0. http://virtual.vtt.fi/inf/pdf/publications/2000/P416.pdf.

Paulk, M.C. (2001) Extreme Programming from a CMM Perspective. IEEE Software, November/December 2001.

Rahikkala, T. (2000) Towards virtual software configuration management. A case study. Technical Research Centre of Finland, Espoo. VTT Publications 409. 110 p. + app. 57 p. ISBN 951-38-5567-8; 951-38-5568-6.
http://virtual.vtt.fi/inf/pdf/publications/2000/P409.pdf

Ronkainen, J. & Abrahamsson, P. (2003) Software development under stringent hardware constraints: do agile methods have a chance? Fourth International Conference on eXtreme Programming and Agile Processes in Software Engineering, XP 2003, Genova, Italy, 25–29 May 2003, Springer-Verlag. Pp. 73–79.

Schamp, A. & Owens H. (1997) Successfully Implementing Configuration Management. IEEE Software, Vol. 14, No. 1, pp. 98–101.

Stevens, R., Brook, P., Jackson, K. & Arnold, S. (1998) Systems Engineering – Coping with Complexity. Prentice Hall, London.

Susman, G. & Evered, R. (1978) An Assessment of the Scientific Merits of Action Research. Administrative Science Quarterly, 1978, 23. P. 582–603.

Taramaa, J. (1998) Practical Development of Software Configuration Management for Embedded System. Technical Research Centre of Finland, Espoo. VTT Publications 366. 147 p. + app. 110 p. ISBN 951-38-5344-6.
http://virtual.vtt.fi/inf/pdf/publications/1998/P366.pdf

Tichy, W. (1988) Tools for Software Configuration Management. In: Winkler, J. (ed.) The German Chapter of the ACM. Vol. 30, International Workshop on Software Version and Configuration Control Grassau, Germany, January 1988. Teubner Verlag, Stuttgart, Germany. Pp. 1–20.

Whitgift, D. (1991) Methods and Tools for Software Configuration Management. John Wiley & Sons, England.

Yin, R.K. (1994) Case study research: Design and methods. 2nd edition. Sage, Newbury Park, CA.

PAPER I

Configuration management support for the development of an embedded system

Experiences in the telecommunication industry

Tools and methods of competitive engineering. Vol. 2. Millpress. The Fifth International Symposium on Tools and Methods of Competitive Engineering (TMCE 2004). Lausanne, CH, 13–17 April 2004. Pp. 605–616.

Proceedings of the TMCE 2004, April 13-17, 2004, Lausanne, Switzerland, Horváth & Xirouchakis (eds.)

CONFIGURATION MANAGEMENT SUPPORT FOR THE DEVELOPMENT OF AN EMBEDDED SYSTEM: EXPERIENCES IN THE TELECOMMUNICATION INDUSTRY

Jukka Kääriäinen
Technical Research Centre of Finland
Finland
jukka.kaariainen@vtt.fi

Jorma Taramaa
Jukka Alenius
Nokia Technology Platforms
Finland
{jorma.taramaa,jukka.alenius}@nokia.com

ABSTRACT

As an embedded system, a radio base station involves several design parties including HW development, HW-related SW development, and application SW development with solutions for air interface standards and the base station itself. The development usually follows the concurrent development paradigm where different SW and HW portions are developed in parallel in separate projects. On the other hand, configuration management (CM) is a widely used and well-known support discipline for product development. This paper reports experiences from the telecommunication industry, in Nokia Networks, related to the embedded systems' CM in a concurrent development environment. It deals with CM concerns that were found especially useful during this study. It also encourages discussion about issues concerning embedded system's CM. The findings are based on an empirical study performed while the authors were working on a project developing DSP application software and hardware-related DSP software. In complex systems the product development is usually organized into separate hierarchical development projects. The paper highlights the efficient inter-project CM planning in a hierarchical project environment. The term inter-project CM planning refers to the planning of those CM practices and constraints that need to be collectively agreed and compatible between projects. Without the inter-project approach the CM might be well realized inside one project but the whole system's CM would be inadequate.

KEYWORDS

Configuration management, concurrent development, embedded system development, DSP software, telecommunication, radio base station

1. INTRODUCTION

Products are getting more complex, containing several implementation technologies such as software (SW) and hardware (HW). These products contain increasingly embedded computer systems, which are application-specific computing devices consisting of standard and custom hardware and software components (ITEA, 1998; MEDEA+, 2002). The development environment is often global and without physical boundaries, consisting of several sites where the team members may come from different cultural backgrounds (Takalo, J. et al., 2000). At the same time, faster time-to-market and better product quality is required, as the companies should be more cost-effective in a harsh business environment.

As an embedded system, a radio base station involves several design parties including HW development, HW-related SW development, and application SW development with solutions for air interface and base station itself (Blyler, J., 2002; Ronkainen, J. et al., 2002). The development usually follows the concurrent development paradigm where different SW and HW portions are developed in parallel in different projects. This paradigm has displaced the traditional sequential design model to meet the very tight time-to-market window. System architects define a system architecture consisting of cooperating system functions that form the basis of concurrent hardware and software design. Although

SW and HW are developed in parallel, the design teams are not isolated from each other, since changes to one part can cause changes to the other (Ernst, R., 1998). For example, HW/SW interface specification, created by a HW design team in cooperation with a SW team, provides the basis for SW driver development. Changes to that specification will likely cause changes to the driver specification. More important than to avoid the change is to adapt to the situation that there will likely be changes. Ernst, R. (1998) states that nowadays, concurrent HW and SW design starts even before the system architecture and specification are finalized. Thus, the proper cooperation between design projects, the communication of changes, consistent understanding of the system, and the management of interface information are essential in order to avoid inconsistencies in the integration phase.

Configuration management (CM) is a widely used and well-known support discipline for product development. The traditional CM process contains the CM elements, such as configuration identification, configuration control, configuration status accounting, and configuration auditing as well as their planning for a project (Taramaa, J., 1998). Reference models and standards include generic models for CM (e.g. in ISO 10007, ISO 12207, SW-CMM (Software Capability Maturity Models) and CMMI (Capability Maturity Model Integration)). Some references, e.g. Taramaa, J. (1998), consider CM especially in the embedded system context.

This paper reports experiences from the telecommunication industry, in Nokia Networks, related to the configuration management of embedded systems in a concurrent development environment. It communicates CM concerns that were found especially useful during this study. The findings are based on an empirical study performed while the authors were working on a project developing DSP application software and hardware-related DSP software. Our perspective on embedded system development is the DSP project and its connections to other SW and HW development projects. The aim is also to open discussion about embedded system's CM. The paper highlights efficient inter-project CM planning. The term inter-project CM planning refers to the planning of those CM practices and constraints that need to be collectively agreed and compatible between projects. Without the inter-project approach the CM might be well realized inside one project but the whole system's CM would be inadequate.

This paper is organized as follows. In the next section, CM concepts are introduced based on literature. In specific, the role of CM in product development and its main activities, the basic terminology, CM organization issues, and automation issues are introduced. The aim is to provide solid background for the basic principles of CM and explain the CM terminology used in this paper. The paper further considers the development of the embedded system in order to expose problem fields for CM. Then the paper reports experiences related to configuration management in embedded systems development in Nokia Networks. Finally, the conclusions and future research needs are identified.

2. CONFIGURATION MANAGEMENT

Configuration management is defined by the IEEE Std 610.12 (1990) as a discipline applying technical and administrative direction and surveillance to identify and document the functional and physical characteristics of a configuration item, control changes to those characteristics, record and report change processing and implementation status, and verify compliance with specified requirements. In short, CM is a discipline controlling the consistency between the parts of an entire system. The roots of CM are in the defense industry as a discipline to resolve problems with poor product quality, parts ordering, and parts not fitting, which were leading to high cost overruns (Berlack, H., 1992). Shamp, A., & Owens, H. (1997) introduce CM as an essential process to increase product quality, development efficiency and enterprise profitability.

Traditional CM and software CM (SCM) are quite similar if they are compared, but SCM and the tools are tailored to the software elements of a system (Bersoff, E. et al., 1980; Gatt, I., & Davidovitz, M., 1990). According to Tichy, W. (1988), SCM differs from CM in the two ways. First, software is easier to change than hardware. Second, SCM is easier to automate. CM in pure SW or HW development is well known, but in complex environments where SW/HW codesign and project hierarchies exist it is more complicated and less examined. SW and HW development do not understood each other. Crnkovic, I. et al., (2003) state that there are cultural differences between SW and HW development, which cause problems in product information management. These problems arise e.g. from different terminology used. In this paper we do not distinguish between SCM and CM, but consider CM planning and organization in general in the embedded system development. We

consider CM from the DSP project perspective including its connections to related HW development and other SW development projects. Even though our viewpoint regarding CM is from the SW project, we will emphasize the integrated planning of software CM and hardware CM to achieve consistent management of a complex product.

The sub-section 2.1 introduces the elements of CM according to literature. Then sub-section 2.2 defines the basic concepts used in CM. After that section 2.3 presents organizational concerns related to CM. The last sub-section 2.4 considers automation issues related to the CM.

2.1. Elements of CM

ISO standards, such as ISO/IEC 10007 (1995) and ISO/IEC 12207 (1995), introduce CM as a *support process for product development*. It is a process of controlling the evolution of complex systems. The CM process contains the basic CM activities and their CM planning (ISO/IEC 12207, 1995; Buckley, F., 1996; Taramaa, J., 1998; Rahikkala, T., 2000). Traditionally CM activities have been divided into configuration identification, configuration control, configuration status accounting, and configuration audit (ISO/IEC 10007, 1995)(Bersoff, E. et al., 1979). Figure 1 depicts the traditional classification of CM elements.

CM planning is used for planning and documenting certain configuration management solution for a project. E.g. how to identify configuration items and control changes. Buckley, F. (1996) views the CM plan as one of the major ways towards communicating a comprehensive understanding of what should be done to maintain the integrity of the products. It provides the means to define CM practices for a project: who is going to do what, when, where and how (Buckley, F., 1996).

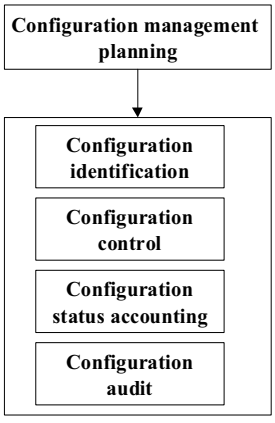


Figure 1 Basic CM elements

Ideally, the whole company can utilize common configuration management. If this can be achieved, the effort spent on the planning, tailoring and modification of configuration management for projects can be avoided. However, the literature emphasizes that CM should be defined for a project and it needs to be maintained, e.g. Buckley, F. (1996), Lyon, D. (1999), and Leon, A. (2000). While no two projects are exactly the same, the CM requirements between different projects also vary (Lyon, D., 1999; Leon, A., 2000; Whitgift, D., 1991). For example, IEEE Std-1042 (1987) states that the size, complexity and criticality of the software system being managed affects the project's CM practices. Whitgift, D. (1991) argues that software CM depends on, for example, the size of the project, project distribution, dependence on third parties software, the item types being developed, and the number of clients. In addition, software and hardware development have their own special needs for CM procedures (Buckley, F., 1996).

Differences between projects do not mean that CM planning should always start from "scratch". Experiences from previous project-specific CM plans can be collected to a generic CM plan or CM "best-practices" for a company, which then will be examined and tailored for future projects (Buckley, F., 1996). This will facilitate the definition of a project-specific CM plan, because a generic CM plan provides "tried and tested", pre-defined structure and information for a project's CM plan creation. This approach provides the possibility for reusing CM practices between projects and assists towards more consistent CM in a company. However, IEEE Std-1042 (1987) states that even if CM is applied as a corporate policy, it should be reexamined each time it is applied in a project to ensure its applicability.

Configuration identification refers to the activities for identifying and documenting configuration items. Configuration control covers activities for controlling changes to configuration items. Configuration status accounting contains activities for formalized recording and reporting of the configuration documents. Configuration auditing refers to the examination to determine whether a configuration item conforms to its configuration documents. In

addition, Buckley, F. (1996) states that an in-process audit (system configuration audit) is performed to determine whether the configuration management process established in an organization is being followed and to ascertain what needs to be improved.

These basic activities have been extended by Dart, S. (1991) to include, for example, SW manufacturing issues (usually referred to as "build") and teamwork issues. On the other hand, IEEE Std-828 (1998) extends basic classification with interface control and subcontractor/vendor control activities in the context of software development. Interface control activities *coordinate changes to the project configuration items' with changes to interfacing items outside the scope of the plan*. According to IEEE Std-828 (1998), for example hardware, system software and other related projects and deliverables should be examined for potential interfacing effects on the project. Subcontractor/vendor control *incorporate items developed outside the project environment into the project configuration items*.

2.2. CM Concepts

ISO/IEC 10007 (1995) defines the term *Configuration Item (CI)* as follows:

Aggregation of hardware, software, processed materials, services, or any of its discrete portions, that is designated for configuration management and treated as a single entity in the configuration management process.

Baseline has an important role when managing changes. It is configuration of a product, formally established at a specific point in time, which serves as a reference for further activities (ISO/IEC 10007, 1995). Baseline is the cornerstone of CM in general. Basically, a baseline is the collection of items (such as documentation and source code) that make up a configuration item at a discrete point in time (Figure 2).

Check-in is the process of moving configuration items into a controlled environment (data vault). On the other hand, *check-out* is the process of copying the item from data vault to the user's area for modification. Usually the CM tool locks the checked out version to prevent concurrent modifications. Check in/out functionalities are basic features of the CM tools.

The term *version* is used to describe the evolution of an item (configuration item's versions). Basically,

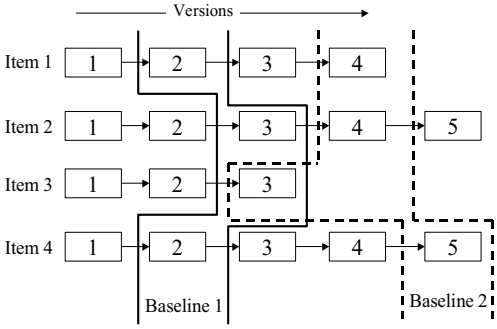


Figure 2 Two baselines from the same items

versions are described as a linear set of developed items (0.0.1 => 0.0.2 => 0.0.3). This is also called revisioning (Whitgift, D., 1991). Linear development is not always possible. Thus we need a concept *branch* (Leon, A., 2000). Here Whitgift, D. (1991) uses the terms: *temporary and permanent item variants*. According to Leon, A. (2000) the branches are deviations from the main development line for the item and they can also be extended from the existing branch. The term *merge* expresses the incorporation of branches (temporary variants) with the main branch. Branch and merge concepts are fundamentals for parallel development (Figure 3).

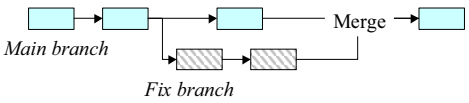


Figure 3 Branching and merging

SW build is the process of generating an executable, testable system from source code. The term *release* can be considered a configuration management action whereby a particular version of software is made available for a specific purpose, e.g. released for test (Buckley, F., 1996).

2.3. CM organization in a company

Leon, A. (2000) argues that detailed procedures and resources are needed for the efficient execution of CM functions. He emphasizes that the most important resource is qualified people. So there is a need for competent persons to do the various CM functions, like CM planning, software building, auditing, etc. However, different organizational structures in different organizations will require the CM team to be structured and positioned differently

(Leon, A., 2000). Leon, A. (2000) introduces three organizational possibilities for software CM:

- Central CM team: Central CM team, operating at a company-level. Strong involvement in the projects' CM issues. This provides good enforcement to generic procedures but might lack the understanding of project-specific needs and project level monitoring.
- Central CM team and project specific CM teams: The central CM team is responsible for generic CM plans and procedures as well as CM system in the company as a whole. The projects' CM teams, operating at a project-level, have the main responsibility for customizing CM plans and procedures for the projects.
- Independent CM teams for projects: Project specific CM teams handle all CM issues in projects. This does not provide coordinated central generic CM planning for a company, but every project takes care of CM issues independently.

The term "CM team" will be used accordingly to denote an organizational support function. In practice, it can be a single person or a full team, depending on the situation (e.g. the size of a project). According to Leon, A. (2000) a project's CM team contains a Configuration Management Officer (CMO), and depending on the project, other technical (e.g. a person who is responsible for release and build activities) and administrative (e.g. secretary) members assigned by the CMO e.g. according to basic CM activities. The CMO is responsible for customizing CM plan and procedures for the project (Leon, A., 2000). He also has reporting responsibility for the central CM team about the project's current CM status. Lyon, D. (1999) considers organization from the multi-technological product's point of view and divides CM organization accordingly into software and hardware CM. He places emphasis on the importance of good communication between the hardware and software elements of CM during the planning phase. This kind of communication enables the definition of CM procedures to support interactions between teams. For example, change impact analysis procedures between hardware and software development teams.

Stevens, R. et al., (1998) argue that in complex systems separate development projects for each subsystem are usually needed. Each project can lead to the creation of further development projects at a level below. Configuration management also occurs in this kind of project environment. Abran, A. & Moore, J. (2001) present that software CM activities take place in parallel with hardware CM activities when software is developed as a part of a larger system containing hardware. They further mention that software CM activities must be consistent with system CM activities. ISO/IEC 12207 (1995) states that the software project's CM plan can also be part of a system CM plan, which contains the whole system's CM support issues. The case of multi-level contracts is similar. The main contractor's CM plan is the main CM plan, and subcontractors will prepare their own plans or include their plans into the main CM plan (ISO/IEC 10007, 1995). ISO/IEC 10007 (1995) further states that these plans, indicating different levels of configuration management, need to be compatible with each other.

2.4. CM automation

The role of CM tools is to support and automate CM functions and provide help for developers. However, Leon, A. (2000) states that CM tools do not solve configuration management problems, but they can be one step towards more effective CM. Even though tools can automate some functions, it is important that the project team knows the CM procedures defined for a project to ensure an understanding of why certain CM functions are needed as well as when and by whom they should be performed. In addition, the project should have instructions on how to use the tools in practice.

There are hundreds of CM tools available containing features depending on their backgrounds and purposes. For example, simple version management tools, CM tools, document management tools, etc. In the context of software engineering so called "SCM tools" are usually used to manage product information. Currently software CM systems provide the following services (Estublier, J., 2000):

- Repository for components.
- Help for engineers' usual activities.
- Process control and support.

Component repository provides basic functionality for storing and distributing product-related information, for example, version management and access control. One characteristic of software CM tools is the ability to handle complex version structures. This means that version management also contains support for the version tree (branching)

where any version can be used to create a new version.

Engineer's support contains support for workspaces (worksets) which are views to a certain set of versioned files for a particular purpose, e.g., development, bug-fixing, or testing. Cooperative work provides the possibility for concurrent item modifications using branching and merging. In addition, SW build support is also an obvious feature because SW designers usually build and test the SW product or sub-system.

Process support provides the means to support a company's pre-defined processes (e.g., product development process, engineering change process, etc.).

The evaluation, selection and implementation of the CM tool are not trivial problems. For example, Schamp, A. (1995) and Dart, S. (1996) have considered the evaluation, selection and implementation issues of the CM tool. Schamp, A. (1995) introduces a selection process that maps key organizational, development, and technical considerations to CM-toolset capabilities. The selection process provides the possibility to rate each tool against its required function and thus support tool selection. Schamp, A. (1995) also states that complex development environments even require an integrated CM toolset. Dart, S. (1996) presents best practices to support the procurement, evaluation, strategic planning and deployment of the CM tool in a company. Dart, S. (1996) divides the scope of a CM solution into an enterprise-wide CM solution and a project-oriented CM solution. In the former case the solution must meet every group's requirements for the CM tool. In the latter case the solution addresses a single project's requirements for the CM tool.

3. EXPERIENCES FROM THE TELECOMMUNICATION INDUSTRY

This section introduces the CM experiences from the field of telecommunication. Our viewpoint to CM is DSP project and its connections to the related SW and HW development. Section 3.1 provides a context for our CM observations. It describes the parties involved into radio base station development, the role of HW and SW development and connections between development parties. It also illustrates the complexity of embedded systems development. Lessons learned have been collected into section 3.2. The intention is to highlight issues, which have been found important during the study and encourage discussion about embedded system's configuration management.

3.1. The development of embedded systems

Embedded systems in radio base station development involve several design parties including HW, driver SW, DSP (Digital Signal Processing) SW, and control processor SW development (Figure 4). The development usually follows the concurrent development paradigm. (Blyler, J., 2002; Ronkainen et. al., 2002).

From the DSP point of view, a crucial activity is SW/HW partition, i.e. SW/HW tradeoff. It divides signal processing functionality solved by algorithms

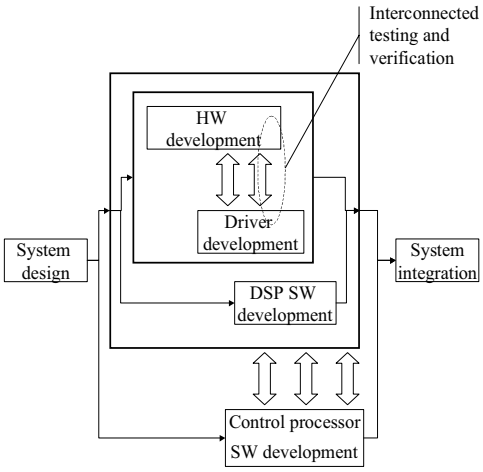


Figure 4 HW and DSP SW codesign related to control processor SW development (adapted according Ronkainen, J. et al., 2002)

into SW and HW. In general, operations with critical execution speed are allocated to HW development as ASICs (Application Specific Integrated Circuit) and further HW components that need field programmability as FPGAs (Field Programmable Gate Array) and operations better suited for SW are allocated for DSP SW development (Kalavade, A. & Lee, E., 1993; Paulin, P. et al., 1997; Gogniat et al., 2000). The increasing use of software in embedded systems enables the inclusion of late specification changes in the design cycle as well as the reuse of

previously designed functions (Goossens, G. et al., 1997). The trend where SW is taking a bigger role for implementing signal processing functionality is known today as SW-defined radio. However, HW-based solutions today hold their own strong position in signal processing -based product solutions and the concurrent SW/HW development practices have to be taken into efficient use. The HW development first operates with abstract descriptions from HW, e.g. VHDL (VHSIC Hardware Description Language) descriptions. After that FPGA and ASIC technology can be used to prototype and implement the design.

In complex systems, the product development is usually organized into separate hierarchical development projects (Stevens, R. et al., 1998). Each project can lead to the creation of further development projects at a lower level. The different parties, such as DSP SW, control processor SW (including Protocol SW and Operation and Maintenance SW) and HW development projects, are not isolated with each other, but detailed interfaces are needed to control the compatibility of units and describe the practical connection between them (Ernst, R., 1998). For example, an HW development project creates HW interface specifications for driver SW development and SW-SW interface specifications are used to describe interconnections between SW units, for example, message interfaces between the DSP and control processor SW. The development of HW related SW happens usually incrementally and thus it actually has some features of agile development (Larman, C. & Basili, V., 2003). Ronkainen, J. & Abrahamsson, P. (2003) has considered the applicability of agile methods for the development of embedded SW.

The management of interfaces and configurations as well as a common understanding about the product plays a central role when managing the product integrity. HW, SW and interface development are closely related. For example, any changes to HW configuration usually cause major changes to the related SW (Blyler, J., 2001). Thus the communication of changes and the management of HW/SW and SW/SW interface information are vital for successful and efficient product development. The lack of communication and configuration management between different design teams and projects will cause difficult, tedious and expensive problems in the system integration phase.

Early verification of HW and SW designs occurs during co-simulation. At the beginning HW is abstracted using e.g. VHDL description and later on FPGA and ASIC prototypes are used to describe final HW. During verification the SW configuration is simulated against the HW configuration (Figure 4). The importance of cooperation during verification is crucial, because SW is used to verify that HW operates properly.

3.2. Lessons learned from the empirical study

The lessons learned from the empirical study are discussed in this section. The observations are made from the DSP SW project's point of view. The observations are divided into general observations and specific CM practice observations:

General Observations

General observations relate to the CM organization and the role of CM in a company as well as its tool support. The following general observations were found important during the study:

Organize and support CM planning in a hierarchical development project:

In complex systems, the product development is usually organized into separate hierarchical development projects. Each project can lead to the creation of further development projects at a lower level. Different parties, such as DSP SW development, control processor-based application SW development, HW development, and system level design and integration teams, are not isolated with each other but coordination and communication is needed during product development, integration and maintenance. CM is usually well realized inside a project but the system CM is usually inadequate and without the clear responsibilities. The projects need to operate with common configuration items, exchange information between projects, and ensure that all changes are properly analyzed in the context of the whole system. This requires compatible inter-project CM practices, clear responsibilities for practices, and a means for data exchange between related projects. Thus the communication and cooperation between projects are already needed during CM planning. Without the inter-project planning approach, the CM is well-realized inside a project but the whole products' CM is inadequate. The following observations were made during the study:

– The use of a company level CM team and project level CM teams is useful because projects have the best knowledge about the sub-systems they are producing. Thus, project level CM teams tailor CM for projects according to the CM plan template and coordinate CM inside the projects. On the other hand, a company level team is needed to improve and generalize CM for the company (e.g. CM plan templates), provide CM consultation and technical support for projects, as well as coordinate overall CM in the company.

– Inter-project CM practices need to be defined for hierarchical projects and their execution should be monitored. This means that CM plans, indicating different levels of configuration management, need to be compatible with each other. According to our experiences, the main project's CM team could be responsible for coordinating the development of inter-project CM activities. To assist in this work the concept of a "CM forum" can be introduced (Figure 5). A CM forum is an inter-project CM planning team containing the main-project's CM team, sub-projects' CM teams, and a representative from a company's CM team. It is a good forum for discussing, communicating and agreeing on common CM practices and responsibilities, and for monitoring (system configuration audit) their usage in hierarchical project. This is also a good channel for exchanging "successful CM practices" between related projects. It is important that the CM forum assemble for a meeting on a regular basis in the CM planning phase. Afterwards, the CM forum can meet less frequently.

Understand the role of CM in a product development project:

The important role of CM as a part of a product development project is not always understood. CM provides support for product development processes. It is not responsible for creating requirements specifications, deciding implementation architecture or developing source code, but it helps to store these configuration items and make them identifiable and manageable.

The role of CM can also be understood too narrowly in a project. It might be seen as an individual-level activity where SW code files are produced, modified and versioned without a broader understanding of change implications. Thus, CM should be seen from a broader perspective especially when several

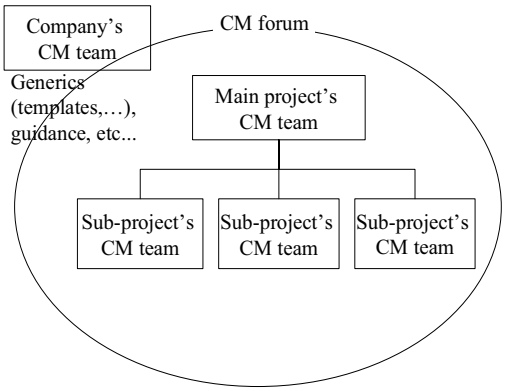


Figure 5 CM forum integrates different teams during inter-project CM planning

geographically distributed development teams exist in a project. The individual and isolated working culture when design files reside in local workstations and changes are not communicated inside a project does not work like a charm anymore.

On the other hand, CM practices can be defined for distributed working, but enforcement fails. Therefore, proper information distribution and the communication of changes fail between development teams and the CM of the whole system is inadequate. According to our experiences it is important that there is consensus in a project about processes, their importance for a project and for product quality, and enforcement for playing according to the rules. Here system configuration audits step into the picture. They are used to determine if the CM process is being followed in a project and to determine what needs to be improved.

Use tools to support CM:

The role of the CM tool is important. They can be used for storing, versioning and accessing configuration items and prevent uncontrolled and parallel modifications to configuration items. However, the CM tool might "dictate" the users to operate according to a certain tool-specific process, which is different than the one in the organization. This can encourage the use of unofficial and uncontrolled procedures. Thus, the processes used in a project are a primary starting point for the tool support. The CM tools should be adapted to support defined processes and project personnel should have a clear understanding of processes. Furthermore the

tool should be flexible enough to be easily adapted when a project's needs for CM support change.

The tools used for SW development, e.g. compiler versions used in projects, should also be under configuration management. Without control some teams might start to use a newer compiler version without notifying other parties about the changes. This leads to a situation where different teams might use incompatible compiler versions that cause problems when teams start to integrate their outputs.

Specific CM Practice Observations

The specific CM practice observations describe those CM activities that were found to be important during the empirical study when producing complex embedded systems in the hierarchical project environment. A common feature of all these observations is that they are all inter-project practices. There has to be clear practices and responsibilities for these activities defined during the inter-project CM planning phase.

Manage common items (manage SW-SW interfaces):

The common configuration items between concurrent projects require consistent and accurate management. These common configuration items include, for example, common header files (e.g. constants, message interfaces) used across several SW projects. Without control these items start to live their own lives. Some projects might modify header files without notifying other projects about the changes. This leads to a situation where different projects might use slightly different header files, which then cause problems during integration.

A single channel (e.g. CM system) is useful for controlling common configuration items. It is used to store and distribute the official compatible set of common items (the baseline of common items). A project can access this repository and select appropriate common items' baseline for their builds. At the same time the project can be confident that other projects also use a compatible set of items. However, using this kind of central point of control needs clear practices. There has to be clear responsibilities and procedures for requesting, evaluating and implementing changes to the items and communicating changes to all parties. This avoids uncontrolled changes to items and several "official copies" from items.

Manage HW/SW co-design (manage HW-SW interfaces and co-verification):

The HW/SW co-design and incremental development approach incrementally produces SW and HW sub-releases, which are simulated against each other. The simulation usually happens concurrently with coding during incremental development (Figure 6).

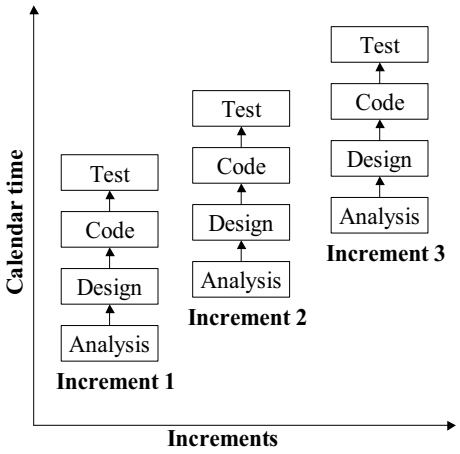


Figure 6 Incremental development model (adapted according Pressman, R., 1997)

Concurrency means that the next increment is under development when the previous increment's release is under simulation. At the beginning HW is abstracted using e.g. VHDL description and later on FPGA and ASIC prototypes are used to describe HW. During verification the SW configuration is simulated against the HW configuration. New versions are produced from SW and HW items in order to meet the requirements of the next increment. However, if there have been problems in simulation, it is essential to be able to trace SW and HW item versions that have been included into the release, to support problem analysis and bug-fixing for the next release.

Another concern here is the management of the HW/SW interface specifications developed by the HW team. Driver SW development uses these specifications, which describe used register interfaces, to create drivers for HW. This specification should be available for the driver development

team and placed under configuration management to avoid uncontrolled changes to it.

In order to support HW/SW co-design a single official channel (e.g. CM system) for managing and delivering specifications and releases between teams is necessary. This prevents the use of unofficial delivery channels, such as network drives, e-mails and floppy disks. The CM tool's component repository –functionality can be used for storing and delivering these items. The valid version from the interface specification should be available for driver development in a specified location in the CM system. Any changes to this specification should be communicated to the relevant parties. Furthermore, the releases should be based on baselines and there should be clear practices on how to identify and document releases to ensure their sufficient traceability. The compatibility of sub-releases during simulation needs to be documented for traceability. The compatibility describes which SW release (ID) has been simulated against certain HW release (ID). If there have been problems in simulation it is then possible to trace the related HW and SW releases back to the CM system for modifications. The modification might need a parallel bug-fixing branch when development and simulation happens concurrently. The bug-fixing of the previous release might happen in the bug-fixing branch, which will be merged into the item's main branch before the next release to ensure that the problems are fixed in the upcoming release.

Manage change hierarchically:

Changes are inevitable in the product development. In hierarchical product development the projects have dependencies and changes in one of the projects can cause changes to the other projects. Thus, change management practices and responsibilities should be defined in such a way that ensures controlled change documentation, evaluation, implementation and notification through project hierarchy and development phases (integrated change management). In particular, the impacts on other projects should be analyzed to avoid inconsistency during integration.

4. CONCLUSIONS AND FUTURE RESEARCH

The study shows that central control for inter-project configuration management is useful in a hierarchical development environment when developing embedded systems. Without the central control the configuration management is well realized inside a project, but the management of the whole system is deficient. Inter-project CM issues already need to be considered during the CM planning phase. CM planning should start from the main project level (system level), which provides CM planning constraints and requirements for CM planning in sub-projects. It also fixes clear responsibilities for inter-project activities such as common items' management, integrated change management, etc. Without central control the inter-project CM activities are rudderless and there are no clear responsibilities, mandated by organization, assigned for these activities. This leads to a situation where CM is well realized inside a project but common CM issues are insufficient. Without clear practices and responsibilities CM problems are likely to occur in the integration, testing and maintenance phases.

The organization has developed practical solutions for supporting embedded systems' CM. However, HW/SW codesign was recognized as the most difficult sector in the study, which especially needs more analysis. The work will continue with the analysis and enhancement of practical solutions for the CM in the company developing complex multidisciplinary products using the concurrent development paradigm.

ACKNOWLEDGMENTS

The authors would like to acknowledge the contributions from several colleagues with Technical Research Centre of Finland (VTT) and Nokia Technology Platform.

REFERENCES

Abran, A., Moore, J., (2001), "SweBok: Guide to the Software Engineering Body of Knowledge. Trial Version 1.0", IEEE Computer Society Press, California.

Berlack, H., (1992), "Software configuration management", John Wiley & Sons.

Bersoff, E., Henderson, V., Siegel, S., (1980), "Software Configuration Management - An Investment in Product Integrity", Prentice Hall.

Bersoff, E., Henderson, V., Siegel, S., (1979), "Principles of software configuration management", Prentice-Hall, Englewood Cliffs, New Jersey.

Blyler, J., (2001), "Challenges Are Ahead For Embedded Software Simulation", Wireless Systems Design. Vol. 6, No. 8, August (2001), pp. 33-34.

Blyler, J., (2002), "Will Baseband Technology Slow Base-Station Evolution?", Wireless Systems Design. Vol. 7, No. 7, July/August (2002).

Buckley, F., (1996), "Implementing configuration management : hardware, software, and firmware", IEEE Computer Society Press, Los Alamitos.

Crnkovic. I., Asklund, U., Persson Dahlqvist, A., (2003), "Implementing and Integrating Product Data Management and Software Configuration Management", Artech House, Boston.

Dart, S., (1991), "Concepts in Configuration Management Systems", In: Feiler P. (ed.): International Workshop on Software Configuration Management (SCM3). Trondheim, Norway, ACM Press, Baltimore, Maryland, pp. 1-18.

Dart, S., (1996), "Best practice for a configuration management solution", In: Sommerville, I. (ed.): Software Configuration management. Lecture Notes in Computer Science, Vol. 1167. International workshop on software configuration management (SCM6), Berlin, Germany, Springer-Verlag, Heidelberg, Germany, pp. 239-255.

Ernst, R., (1998), "Codesign of Embedded Systems: Status and Trends", IEEE Design & Test of Computers, Vol. 15 No. 2, April-June (1998), pp. 45 –54.

Estublier, J., (2000), "Software Configuration Management: A Roadmap", Ed. Anthony Finkelstein, The Future of Software Engineering, 22nd International Conference on Software Engineering (ICSE 2000).

Gatt, I., Davidovitz, M., (1990), "Configuration management-integration of software and hardware at top-level Management", CompuEuro '90. Proceedings of the 1990 IEEE International Conference on Computer Systems and Software Engineering (1990), pp. 532 – 535.

Gogniat, G., Auguin, M., Bianco, L., Pegatoquet, A., (2000), "A Codesign Back-End Approach for Embedded System Design", ACM Transactions on Design Automation of Electronic Systems, Vol. 5, No.3, July (2000), pp. 492-509.

Goossens, G., Van Praet, J., Lanneer, D., Geurts, W., Kifli, A., Liem, C., Paulin, P., (1997), "Embedded Software in Real-Time Signal Processing Systems: Design Technologies", Proceedings of the IEEE , Vol. 85, No. 3, March (1997), pp. 436 –454.

IEEE Std-610.12, (1990), "IEEE standard glossary of software engineering terminology".

IEEE Std-828, (1998), "IEEE Standard For Software Configuration Management Plans".

IEEE Std-1042, (1987), "IEEE guide to software configuration management".

ISO/IEC 10007, (1995), "Quality management – Guidelines for configuration management", (1995), International Standard.

ISO/IEC 12207, (1995), "Information technology - Software life cycle processes", (1995), International Standard.

ITEA – Information Technology for European Advancement, (1998), Rainbow Book, Eindhoven University, Netherlands.

Kalavade, A., Lee, E., (1993), "A hardware-software codesign methodology for DSP applications", IEEE Design & Test of Computers, Vol. 10, No. 3, March (1993), pp. 16 –28.

Larman, C., Basili, V., (2003), "Iterative and incremental development: a brief history", IEEE Computer, Vol. 36, No 6, June 2003, pp 47 –56.

Leon, A., (2000), "A Guide to software configuration management", Artech House, Boston.

Lyon, D., (1999), "Practical CM - Best Configuration Management Practices for the 21st Century" (2nd edition), RAVEN Publishing Company.

MEDEA+ - "Design Automation Roadmap", (2002), The MEDEA+ Office, Paris, France.

Paulin, P.G., Liem, C., Cornero, M., Nacabal, F., Goossens, G., (1997), "Embedded Software in Real-Time Signal Processing Systems: Application and Architecture Trends", Proceedings of the IEEE, Vol. 85, No. 3, March (1997), pp. 419-434.

Pressman, R., (1997), "Software engineering: a practioner's guide", McGraw-Hill.

Rahikkala, T., (2000), "Towards Virtual Software Configuration Management: a case study", Technical Research Centre of Finland. VTT Publications 409, Espoo.

Ronkainen, J., Abrahamsson, P., (2003), "Software development under stringent hardware constraints : do agile methods have a chance?", Fourth International Conference on eXtreme Programming and Agile Processes in Software Engineering, XP 2003, Genova, Italy, 25 - 29 May 2003, Springer-Verlag, pp. 73 - 79

Ronkainen, J., Taramaa, J., Savuoja, A., (2002), "Characteristics of Process Improvement of Hardware-related SW", LNCS 2559: Product Focused Software Process Improvement, 4th International Conference on Product Focused Software Process Improvement, PROFES 2002, Rovaniemi, Finland, December 2002. Springer-Verlag. Berlin Heidelberg, pp. 247 – 257.

Schamp, A., (1995), "CM-tool Evaluation and Selection", IEEE Software, Vol. 12, No. 4, pp. 114 –119.

Schamp, A., Owens H., (1997), “Successfully Implementing Configuration Management”, IEEE Software, Vol. 14, No. 1, pp. 98 –101.

Seppänen, V., (1990), “Acquisition and Reuse of Knowledge to Design Embedded Software”, Technical Research Centre of Finland. VTT Publications 66. Espoo.

Stevens, R., Brook, P., Jackson, K., Arnold, S., (1998), “Systems Engineering -Coping with Complexity”, Prentice Hall, London. UK

Takalo, J., Taramaa, J., Savolainen, P., Partanen, J., (2000), “Experiences of Distributed Product Data Management of Electro-mechanical Products in Multisite Organization”, Proceeding of the Third International Symposium on Tools and Methods for Competitive Engineering - TMCE 2000, Delft University Press, Netherlands, pp. 217-224.

Taramaa, J., (1998), “Practical Development of Software Configuration Management for Embedded System”, Technical Research Centre of Finland. VTT Publications 366. Espoo.

Tichy, W., (1988), “Tools for Software Configuration Management”, In: Winkler J. (ed.): the German Chapter of the ACM Vol. 30, International Workshop on Software Version And Configuration Control, Grassau, Germany, January 1988. Teubner Verlag, Stuttgart, Germany. pp. 1-20.

Whitgift, D., (1991), “Methods and Tools for Software Configuration Management”, John Wiley & Sons, UK.

PAPER II

Improving software configuration management for extreme programming

A controlled case study

EuroSPI 2003 Proceedings. Verlag der technischen Universität Graz, European Software Process Improvement, EuroSPI'2003. Graz, Austria, 10–12 Dec. 2003. 10 p.

Improving Software Configuration Management for Extreme Programming: A Controlled Case Study

Juha Koskela, Jukka Kääriäinen and Juha Takalo
Technical Research Centre of Finland, VTT Electronics
P.O.Box 1100, FIN-90571 Oulu, Finland
{Juha.Koskela, Jukka.Kaariainen, Juha.Takalo}@vtt.fi

Abstract

Extreme programming (XP) is currently the most popular agile software development method. It is as its best for small teams developing software subject to rapidly changing requirements. Software configuration management (SCM) is a method of bringing control to the software development process. SCM is known as an indispensable activity that must take place whenever developing software. It is inseparable part of quality-oriented product development regardless of development method. Existing studies show that SCM is partially addressed via XP's collective ownership, small releases, and continuous integration practices. However, currently there exist very few empirical data on SCM exploitation in XP. This paper reports results from a controlled extreme programming case study supported by well-defined SCM activities and tools. Results show that SCM activities and tools, when properly used, provide essential support for XP development process and its practices.

Keywords

Agile methods, extreme programming, software configuration management

1 Introduction

Extreme programming (XP) developed by Kent Beck is currently the most popular agile method [e.g. 1,2]. XP like other agile methods focus on generating early releases of working products. They aim to deliver business value immediately from the beginning of the project. Software configuration management (SCM) is a method of bringing control to the software development process and has been proved to be an invaluable part of developing high quality software [3]. It is also known as an indispensable activity that must take place whenever developing software [4] and, therefore, it can be seen important also in XP and other agile methods. Paulk [5] presents that SCM is partially addressed in XP via collective ownership, small releases, and continuous integration. However, traditional definition divides SCM into configuration identification, configuration control, configuration status accounting and configuration audits [6, 7]. Thus, XP's approach to SCM can be seen implicit and incomplete.

Currently there exist very few empirical data of SCM exploitation in XP. This paper reports results from a controlled extreme programming case study supported by SCM. A team of four developers was acquired to implement a system for managing the research data obtained over years at a finish research institute. As the project team had no earlier experience on XP or SCM, they were given two days practical training before the start of the project. The focus in this paper is on reporting the SCM experiences and observations found during the project. In this case software configuration management implementation was taken into account right from the beginning of the project. Results show that SCM activities and tools, when properly used, provide essential support for XP development process and its practices.

The paper is organized as follows. The following section introduces the XP method, software configuration management and related research. This is followed by a description of the research methods, research settings, the results and the discussion. Lastly, section six concludes the paper.

2 Related Research

This section briefly introduces Extreme Programming method and software configuration management. In addition also the current knowledge of SCM in XP is presented.

2.1 Extreme Programming

Extreme programming (XP) developed by Kent Beck is an agile method for teams developing software subject to rapidly changing requirements. It is focused on delivering immediate business value to the customers. The XP process can be characterized by short development cycles, incremental planning, continuous feedback, reliance to communication and evolutionary design [8]. According to Beck [9] rather than planning, analyzing, and designing for the far to the future, XP suggests to do all of these activities at a little at a time, throughout software development.

Primarily, XP is aimed at object-oriented projects using at most dozen programmers in a one location [10]. According to surveys, XP is currently the most popular and well-known method in the agile family of methodologies [e.g. 1, 2]. It is made up of a simple set of common-sense practices. In fact, most of XP ’s practices have been in use for a long time and therefore they are not unique or original. Many of them have been replaced with more complicated practices in the process of time, but in XP they are collected together. Practices are planning game, small releases, metaphor, simple design, testing, refactoring, pair programming, collective ownership, continuous integration, 40-hour week, on-site customer, coding standards, open workspace and just rules. From the viewpoint of our study, the most interesting practices are collective ownership and continuous integration. Collective ownership means that anyone can change anything at any time. Respectively, continuous integration recommends to integrate changes often with existing code. For an overview of other agile methods readers are re-

ferred to [e.g. 11].

2.2 Software Configuration Management

ISO standards [6, 12] introduce configuration management (CM) as a support process for a product development. It is a process of controlling the evolution of complex systems. CM process includes elements containing the basic CM activities and their CM planning [12, 13, 14, 15]. Traditionally CM activities have been divided into configuration identification, configuration control, configuration status accounting and configuration audit [6, 7].

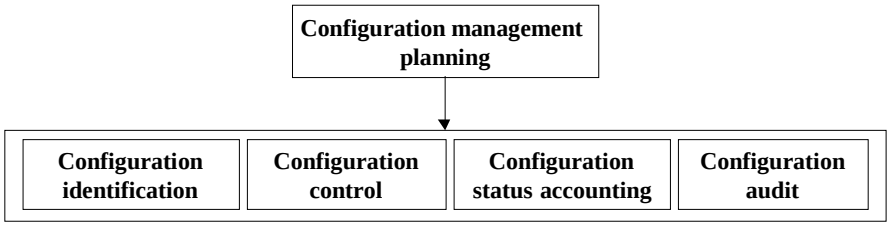


Figure 1. Basic CM elements.

CM planning is used for planning and documenting certain configuration management solution for a project. Buckley [13] views CM plan as one of the major ways to communicate comprehensive understanding on what should be done to maintain the integrity of the products. CM planning provides means to define CM practices for a project: who is going to do what, when, where and how [13]. Standards such as IEEE Std-828 [16] and ISO 10007 [6] provide recommendations for CM plan contents. These recommendations can be utilized when creating CM plans for a company.

The role of CM tools is to support and automate CM functions and provide help for developers. However, Leon [17] states that CM tools do not solve configuration management problems, but they can be one step towards more effective CM. Even though tools can automate some functions, it is important that project team knows CM procedures defined for a project to ensure understanding why certain CM functions are needed as well as when and by whom they should be performed.

2.3 SCM in XP

Currently, there exist very few studies of software configuration management in XP. Paulk [5] has reviewed XP from the perspective of the Capability Maturity Model (CMM) and presents that SCM is partially addressed via collective ownership, small releases, and continuous integration. Christensen [3], who has researched change tracking in rapid and extreme development, presents that SCM has something to offer fast-paced development processes like XP. Our point of view is similar to Christensen as we think that SCM is needed in XP and it can support the XP development process.

XP literature emphasizes the importance of SCM automation to support XP practices [e.g. 10, 18 and 19]. Jeffries et al. [10] state that, in general, SCM tool should be easy to use. Further they emphasize that there should be as few restrictions as possible in SCM tool. For example, no passwords, no group restrictions, as little ownership hassle as possible. They mention SCM tools such as ENVY, CVS and Visual SourceSafe that can be used in XP projects. Succi and Marchesi [19] present that XP practices, such as collective ownership and continuous change integration, are not particularly well supported by traditional version and configuration management systems. Therefore, they have developed a new paradigm for supporting team software development, called *team streams*, that provides dynamic and easy to use team support. Succi and Marchesi [19] enumerate the team streams' characteristics that make them well suited for XP, such as easy to learn and to use, continuous integration, collective code ownership, fully optimistic concurrency, conflict detection and merging and tightly integrated team support.

Bendix and Hedin [18] report how CVS is used by students for simple configuration management on XP projects. According to their results, the students found CVS to be indispensable for the success of the group's effort. However, empirical data of comprehensive SCM exploitation in XP is rare and it indicates that more studies are required.

3 Research Approach

The purpose of this section is to clarify research methods used, and to introduce the basis for the research.

3.1 Research Methods

We used literature study as basis for our research and adopted principles from case study. Literature study was carried out to reveal the current state of software configuration management in extreme programming and to assist us in defining the configuration management principles and technical support for them. Research itself was carried out as a case study.

Järvinen [20] and Yin [21] presents the characteristics of a case study. Case study, as well as controlled experiment, uses research questions like how and why and it focuses on contemporary events. The difference between controlled experiment and case study is that experiment requires the control over behavioral events but the case study does not require such a control [21]. According to Järvinen [20], one specific aspect for controlled experiment is that researcher should be "a neutral observer" when the experiment is carried out in a laboratory environment. In our case, a researcher was in a role of customer who is an active participant in XP based product development and has a control to experiment through the required product features. Therefore, our research can be seen as a case study instead of controlled experiment. The framework, i.e. research manuscript, was created to guide us throughout the experiment. To decrease the number of data needed to collect during the case study the research focus was defined beforehand as the principals of case study emphasizes. The other factors we defined for the experiment were product features, product development environment and procedures, and templates for data gathering and SCM purposes.

3.2 Research Settings

A team of four developers was acquired to implement a system for managing the research data obtained over years at a finish research institute. The developers were 4-6th year students with 1-5 years of industrial experience in software development. Because team members had no earlier experience of XP nor software configuration management, they were given one day training on XP and other on SCM. Before actual training team members studied two books on XP to get the basics for the training. Both training days included theoretical and practical parts. Theoretical part of SCM training dealt with main SCM activities and their organization during the project. Practical part was focused on SCM tool usage in the XP environment. The project was conducted in 1-2 weeks iterations total of two months work effort. At the end of every iteration project team had produced full working software release, which was given to 17 allocated testers for the purpose of system testing. Table 1 shows the technical environment used in the development of system.

Table 1. Technical implementation environment.

Item	Description
Language	Java (JRE 1.4.1), JSP (2.0)
Database	MySQL (Core 4.0.9 NT, Java connector 2.0.14)
Development Environment	Eclipse (2.1)
SCM	CVS (1.11.2); integrated to Eclipse
Unit testing	JUnit (3.8.1); integrated to Eclipse
Documents	MS Office XP
Web Server	Apache Tomcat (4.1)

Planning of SCM implementation was conducted in the beginning of the first iteration using the generic SCM plan, which already included some general information and served as a template for project specific SCM plan. The generic SCM plan template was done according to IEEE standard 828-1998 [16]. Tailoring of generic plan template included adding of roles and responsibilities, schedules and project specific SCM practices.

The role of SCM tool in XP was found important in the literature study and, therefore, it was considered very carefully. The chosen tool was CVS mostly because the development environment (Eclipse) was shipped with a built in client for the CVS. In addition, CVS also supports XP's collective ownership and continuous integration -practices.

Traditional change management approaches can be seen too rigid for XP. Therefore, the purpose was to create simple change management approaches for the needs of this project including both release change management and customer change management processes. Basis for the release change management process was that team should be empowered to make changes through comprehensive unit testing according to collective ownership and continuous integration -practices. Table 2 shows the six stepped development process that was supported by CVS. The purpose of these steps was to ensure that developers would integrate only working versions of software to the repository.

Table 2. Release change management process.

1. Implement a task using the test first methodology
2. Compile and run all unit tests
3. Repeat the first two steps until the task is finished and all unit tests run at 100 %
4. Synchronize with repository
5. If conflicts or incoming files merge and go back to step 2
6. If no conflicts or incoming files commit changes to the repository

The basis for the customer change management process was that in XP the customer decides what to change and s/he can change the requirements at any time [8]. Thus, customer maintained a simple change request form (Excel form), in which change requests were listed. Figure 2 shows that the customer served as a filter between testers and project team. Testers send feedback to customer and he removed, for example, duplicates and impractical proposals. Every change request was equipped with additional information like classification, description and priority of change. New change request were approved or disapproved in the next iteration's planning game. As mentioned customer decides what to change, but project team helped by bringing their technical knowledge to decision-making. This simple customer change management process was mobilized after the first product baseline had been approved and the system had been sent to testers.

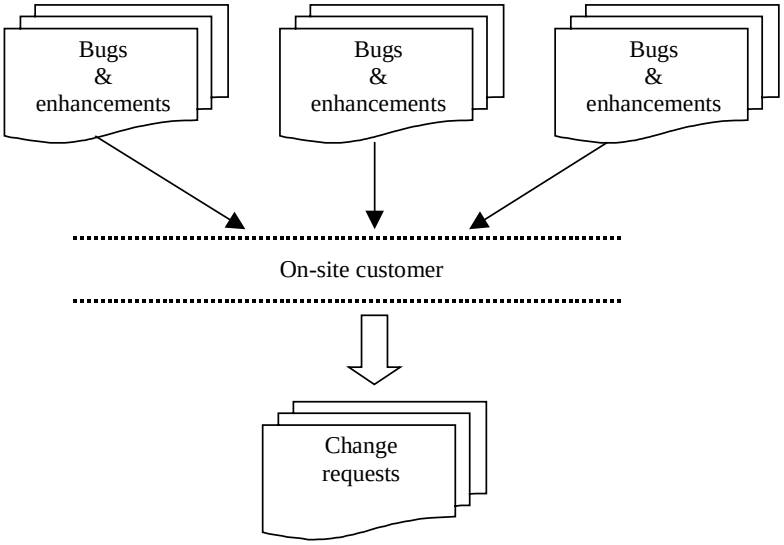


Figure 2. Customer change management process.

Three types of audits were performed during the project. Functional (FCA) and physical configuration audits (PCA) were conducted using a simple checklist method containing a number of requirements of both FCA and PCA. This checklist was examined at the end of every development cycle. FCA's requirement was, for example, that built software system corresponds with the user stories. The third type of audits, in-process audits, were conducted to ensure that SCM practices were followed as planned and everything was working correctly.

4 Results

The focus of this section is on reporting the SCM experiences and observations found during the project. This includes both quantitative and qualitative data of the project. First three development cycles were two-week releases and the following two were one-week releases. The release number six was two days long and included final bug fixes and enhancements of the system. Table 3 shows the data obtained from the project's releases. Data was drawn from the CVS tool.

The essential role of SCM tool emphasized during the project. Development artifacts were continuously safe in CVS server and developers integrated their changes through well-defined process. Team integrated code changes an average of two times per hour. Development team was interviewed after the project had been finished and asked that how often the code should be integrated. According to their answers, team tried to integrate at least once during the task, but even more often if possible.

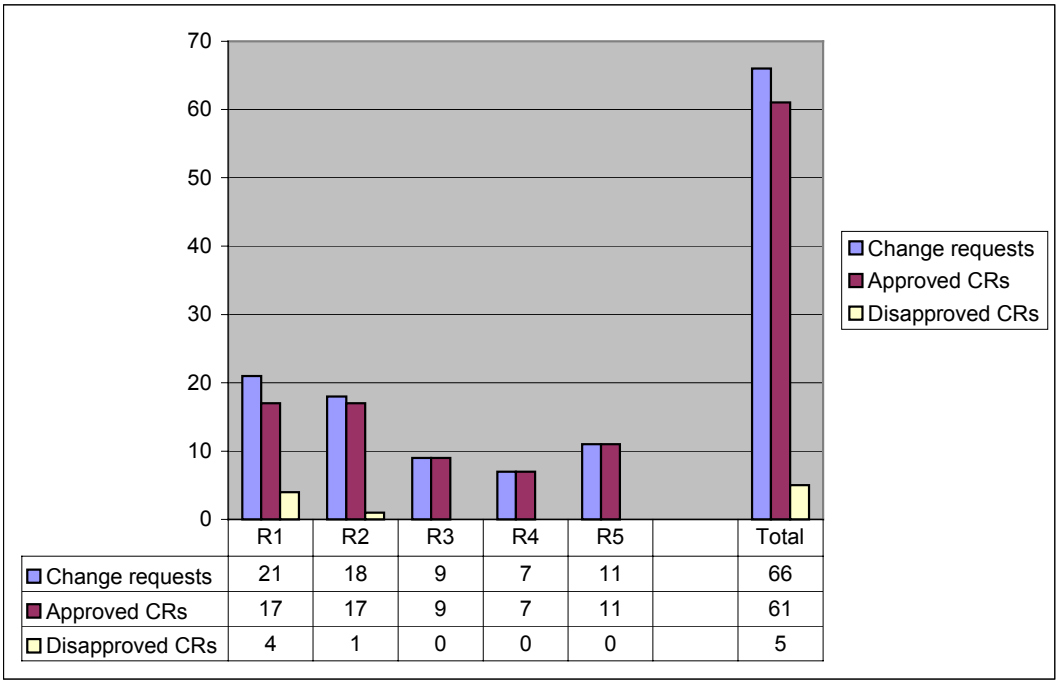
There was couple of times during the project, when intervals between integrations stretched too long and lots of changes directed to the same files. One reason for this was that developers wanted not to integrate unfinished items to the repository. Regardless of the reason, not following of continuous integration -practice caused complex merge-operations. Usually changes of one code integration cycle focused to a one file. However, the average number of files per code integration was 2.6. Due to CVS, team was able resume any earlier version of development artifacts they needed. There were situations when problems with item under development could not be solved without comparing the current item version with the previous ones. However, generally only the last versions of items in the repository were relevant.

Table 3. Concrete data from releases.

Collected data	Rel. 1	Rel. 2	Rel. 3	Rel. 4	Rel. 5	Rel. 6	Total
# Code integrations	65	81	71	42	41	17	317
Code integrations/workingday	8.1	10.1	7.9	10.5	8.2	8.5	8.9

Avg. time between two code integrations (min)	26	21	40	31	27	30	29
Avg. number of files per code integration	1.7	2.4	3.1	2.6	3.0	3.0	2.6
# New files (code)	22	15	16	1	9	0	63
# New files (other)	3	3	18	20	2	0	48
# New versions (code)	111	198	219	110	123	49	810
# New versions (other)	7	9	30	58	14	1	119

Both the release change management and customer change management processes proved to practical for this project. However, this was not so straightforward as both processes required commitment from the persons performing. The six stepped release change management process (see Table 2) ensured that everything in the CVS repository was always 100 % working. The basis was that release change management process steps were carefully defined and the SCM tool supported the process. In addition developers had to strictly follow the process steps and make unit tests comprehensive enough. Respectively the customer change management process required commitment from the on-site customer. Figure 3 shows data of releases' change requests. The on-site customer filtered testers' feedback to a list of change requests. As we can see from Figure 3, some of the change requests in first two releases had to disapprove, because they would have been too complicated to implement within the project's time schedules. A one example of disapproved change requests was version control feature. Approved change request were written down to tasks during the planning game and scheduled as a part of next release's content.



	R1	R2	R3	R4	R5		Total
Change requests	21	18	9	7	11		66
Approved CRs	17	17	9	7	11		61
Disapproved CRs	4	1	0	0	0		5

Figure 3. Data of releases' change requests.

Functional and physical configuration audits were conducted at the end of development cycles total of 5 times. If the results reviewed satisfied the audit requirements, system was ready to be released. Not once audit results were rejected. However, 2/5 times of FCAs and PCAs the results were acceptable with changes. These were the times when coding standards was followed insufficiently. SCM system audits were conducted at the beginning of every development cycle (except the first one). Therefore, the total count for in-process audits was four. In practice only two first of in-process audits produced changes to SCM practices documented in SCM plan. These changes concerned, for example, roles and responsibilities and change management practices.

In the final interview project team were asked that how important they see SCM in this kind of projects. Answers were very similar and emhasized the importance of SCM. SCM tool support was seen indis-

pensable. The most experienced team member answered as follows:

> *"It (SCM tool) is an essential part in the support of teamwork and should be axiomatic in every project."*

5 Discussion

The results presented in the previous section pointed some important requirements for SCM implementation in XP. Bendix and Hedin [18] have reported that students considered the merge support extremely helpful in XP projects. During this study there were only four persons (two pair programming pairs) developing the system. Results show that development team integrated their code changes on an average of 9 times per working day. XP literature [e.g. 8, 10] do not give exact integration intervals, but suggest to integrate often. Despite the fact that team integrated their changes often, merge operations occurred regularly, almost daily. Therefore, from SCM tool perspective it is important that there is straightforward and easy to use merge support, because concurrent changes are likely to happen. Bendix and Hedin [18] also report that very few of students in their XP projects had ever retrieved an older version of file. The results of this study support their findings, because generally only the last versions of files in the repository were relevant. However, there were some exceptions when previous versions were needed and, therefore, it is important that previous items and releases are identified and traceable. The great number of files and their versions speak for importance of version management. Overall SCM tool should be easy to use and not disruptive to encourage developers in tool exploitation. Leon [17] has argued that the role played by SCM tools is becoming more and more important in today's complex software development environments. In this study students found SCM tool very important and also the results show that SCM tool had an important role in this project. CVS proved to be good choice for project's SCM tool. Development artifacts were continuously in safe and developers had no fear to implement and integrate their changes.

SCM audits revealed, for example, that coding standards were not always completely followed. Therefore, audits had a positive influence to the internal quality of software releases. Results show that in practice only two first of in-process audits produced changes to SCM practices documented in SCM plan. This indicates that in-process audits were essential so that SCM practices matured to the level, where they were viable for this project.

Jeffries et al. [10] suggest to write problem reports to cards and schedule them in a current iteration or future iterations. Basically our customer change management process was based on their approach. However, our solution contained also change filtering, documentation and evaluation phases. Then, according to Jeffries at al.'s [10] suggestion, approved change requests were written down to task cards by development team and scheduled to the release. The results of this study show that this definite customer change management process works if the on-site customer have time to commit.

6 Conclusions

Agile software development methods have attracted great attention in the last few years. XP, currently the most well known agile method, is focused on delivering immediate business value to the customers. SCM is a method of bringing control to the software development process and is known as an inseparable part of quality-oriented product development regardless of development method. Thus, the value of SCM should not be underestimated in the case XP and other agile methods. Current studies show that SCM is partially addressed in XP via collective ownership, small releases, and continuous integration. However, traditional definition divides SCM into configuration identification, configuration control, configuration status accounting and configuration audits. Currently there exist very few empirical data of SCM exploitation in XP. This paper reports results from a controlled extreme programming case study supported by well-defined SCM activities and tools. Project team was trained to SCM before the start of the project. The SCM implementation was taken into account right from the beginning of the project, which means that project team tailored generic SCM plan template for the purpose of this project. During the project SCM implementation was audited regularly to ensure that SCM practices were followed as planned and everything was working correctly.

Results show that especially SCM tool has a remarkable role in XP project. Easy to use SCM tool with well-defined release change management process enables that project team can develop the system according to XP's collective ownership and continuous integration -practices. In other words the SCM tool can support the use of these practices. The results also show that SCM tool should have straightforward merge support and previous versions of files need to be identified and traceable. However, SCM tool was only a one part of project's SCM implementation. A simple customer change management process proved to be practical, but it was find out that on-site customer's commitment is required. In-process audits were found essential in order that project's SCM practices can be modified or even removed if needed. Because of regular in-process audits, SCM plan is up to date in every release. Functional and physical configuration audits had a positive influence to the internal quality of software releases. Overall the results show that SCM activities and tools provide essential support for XP development process and its practices.

7 Literature

[1] Maurer, F., Martel, S.: Extreme Programming: Rapid Development for Web-Based Applications. IEEE Internet computing. Vol. 6, Issue 1, 86-90 (2002)

[2] Charette, R.: The Decision Is in: Agile Versus Heavy Methodologies. Cutter Consortium e-Project Management Advisory Service, Vol. 2, no. 19 (2001)

[3] Christensen, H. B.: Tracking Change in Rapid and eXtreme Development: A Challenge to SCM-tools?. Tenth International Workshop on Software Configuration Management (2001)

[4] Compton, S. P., Conner, G. R.: Configuration Management for Software. New York: Thomson Publishing company (1994)

[5] Paulk, M. C.: Extreme Programming from a CMM Perspective. Paper of XP Universe (2001)

[6] Quality management – Guidelines for configuration management. International Standard. ISO/IEC 10007:1995. (1995)

[7] Bersoff, E., Henderson, V., Siegel, S.: Software Configuration Management - An Investment in Product Integrity, Prentice Hall (1980)

[8] Beck, K.: Extreme Programming Explained: Embrace Change. Reading, MA.: Addison Wesley Longman, Inc. (1999)

[9] Beck, K.: Embracing Change with Extreme Programming. Computer. Vol. 32, no 10, 70-77 (1999)

[10] Jeffries, R., Anderson, A., Hendrickson, C.: Extreme Programming Installed. NJ: Addison-Wesley (2001)

[11] Abrahamsson, P., Salo, O., Ronkainen, J., Warsta, J.: Agile Software Development Methods: Review and Analysis. Technical Research Centre of Finland, VTT Publications 478 (2002)

[12] Information technology - Software life cycle processes. International Standard. ISO/IEC 12207:1995. (1995)

[13] Buckley, F.: Implementing configuration management : hardware, software, and firmware. IEEE Computer Society Press, Los Alamitos (1996)

[14] Taramaa, J.: Practical devalopment of software configuration management for embedded system. Technical Research Centre of Finland. VTT Publications 366. Espoo (1998)

[15] Rahikkala, T.: Towards virtual software configuration management: a case study. Technical Research Centre of Finland. VTT Publications 409, Espoo, (2000)

[16] IEEE Standard For Software Configuration Management Plans. IEEE Std-828-1998. (1998)

[17] Leon, A.: A Guide to software configuration management. Artech House, Boston (2000)

[18] Bendix, L., Hedin, G.: Summary of the subworkshop on extreme programming. Nordic Journal of Computing 9 (2002)

[19] Succi, G., Marchesi, M.: Extreme Programming Examined. Boston: Addison-Wesley (2001)

[20] Järvinen, P.: On Research Methods. Tampereen Yliopistopaino Oy, Juvenes-Print, Tampere (2001)

[21] Yin, R., K.: Case study research: Design and methods, Sage Publications, Beverly Hills Ca (1989)

8 Author CVs

Juha Koskela

Juha Koskela has now worked since June 2002 as a research trainee and since July 2003 as a research scientist in Embedded product data management -research group at the Technical Research Centre of Finland. He has received Master of Science Degree in June 2003 in information processing science from the University of Oulu. His research interests include Software Configuration Management and agile software development methods.

Jukka Kääriäinen

Jukka Kääriäinen has now worked since 1999 as a research scientist in Embedded product data management -research group at the Technical Research Centre of Finland. He has received Master of Science Degree in 1999 in Computer Science and Bachelor of Science Degree in 1994 in Industrial Engineering and Management. His research interests include Software Configuration Management and Product Data Management (PDM).

Juha Takalo

Juha Takalo has received Master of Science Degree in computer science in 1995 from the University of Oulu. He works as a research scientist in Embedded product data management -research group at the Technical Research Centre of Finland. His current research interests cover product data management (PDM), configuration management (CM), and document management.

PAPER III

Supporting requirements engineering in extreme programming

Managing user stories

Proceedings of the ICSSEA 2003, 16th International Conference, Software Systems Engineering and their Applications. Vol. 4. Paris, FR, 2–4 Dec. 2003, ICSSEA. 8 p.

Supporting requirements engineering in extreme programming: managing user stories

Jukka Kääriäinen, Juha Koskela, Juha Takalo, Pekka Abrahamsson, Kari Kolehmainen

VTT Technical Research Centre of Finland
P.O.Box 1100, FIN-90571 Oulu, Finland
Phone: +358 8 551 2191, Fax: +358 8 551 2320
{jukka.kaariainen, juha.koskela, juha.takalo, pekka.abrahamsson}@vtt.fi, kari.kolehmainen@annuminas.com

Abstract

One objective for the agile methods is to lower cost of changing requirements. Currently the most popular agile software development method is Extreme Programming (XP). XP addresses this issue by simplifying management tasks and documentation while the traditional software engineering places more emphasis on strict control and extensive documentation. Requirements management (RM) is the activity that ensures that requirements are traceable and all changes to requirements are properly handled. In a dynamic and fast moving project with an iterative process, RM may tie up too much resources. Requirements management and configuration management (CM) are only implicitly addressed by XP. It may be that the method developers viewed RM and CM too bureaucratic, heavy weight or ceremonial to be included in XP. Currently, in the XP process, user requirements called as "user stories" are written and managed on paper cards. The objective of this paper is to examine the challenges involved in the requirements management in an XP project. The aim is also to study possibilities to integrate the support for user story management into single tool framework. Based on the study, an approach for managing user stories with the respect of XP's basic principles is depicted. Our study is based on an empirical XP case study. According to our study, the following issues were recognized especially important when considering the management of requirements in XP. First, XP provides the basic set of practices that should be shaped into the development situation. This means that practices are tailored for the purposes of a project and the organization, and they can be further changed on-the-fly on periodic process assessments, i.e. if they do not work. Thus, the tool support should not force detailed procedures but provide just enough basic abilities for storing, relating and retrieving user stories and tasks. Second, the tool should allow the XP process to remain agile. This means that the tool should not jeopardize XP's intentions for open communication and lightweight management and documentation. Third, requirements management tool support should be integrated into the project's overall development environment. This allows the project team to operate via one channel from a user story definition, through implementation, up to testing. Our solution for the management of user stories and tasks is called StoryManager. The solution has been integrated as plug-in into Eclipse –tool integration framework to enable integrated environment for an XP project.

Keywords

Extreme programming, requirements management, requirements engineering

1 Introduction

New software development methodologies called as agile methods have been developed to address the needs for lightweight and faster software development processes [3]. Currently the most popular one is Extreme Programming (XP), an agile development method developed by Kent Beck [4]. One objective for the agile methods is to lower cost of changing requirements. XP addresses this issue by simplifying management tasks and documentation while the traditional software engineering places more emphasis on strict control and extensive documentation. In order to achieve simplicity, XP uses an iterative and incremental software process and very short development cycles. Further, it minimizes documentation and ties up customer involvement into the product development.

Requirements management (RM) ensures that requirements are traceable and all changes to requirements are properly handled [20]. In the development of complex software products, the requirements management can be difficult and effort consuming when detailed traceability is targeted. In a dynamic and fast moving project with an iterative process, this may tie up too much resources. XP strives for efficient use of resources, an early introduction of functional product releases and continuous feedback from the on-site customer. Therefore, it is inefficient if major proportion of resources has to be used for management tasks that do not deliver concrete results.

The objective of this paper is to examine the challenges involved in the requirements management in an XP project. The aim is also to study possibilities to manage user stories and tasks electronically as part of integrated tool framework. Requirements engineering, especially capturing and analyzing user's conception about the system, happens in planning game phase in XP. Thus, our approach focuses on planning game phase of the XP process. First, the paper introduces XP and RM concepts. After that, the related work concerning user story management is considered. It describes also how user requirements are handled in the current XP process and identifies the potential problem areas that arise from the use of manual solution. Then, findings based on an empirical XP case study are introduced. Based on these findings, an approach for managing user stories with the respect of XP's basic principles is depicted. Finally, conclusions and future research activities are identified.

2 Background

The following sub-sections introduce XP and RM concepts as well as related work.

2.1 Extreme Programming (XP)

Extreme programming (XP) as a concept has emerged in the late 90's along with Kent Beck's book "Extreme Programming explained: Embrace Change" [4]. Along with XP, many more of these "agile" methods have emerged (for an overview see e.g., [2]). XP addresses issues of changing requirements and their cost by simplifying management tasks and documentation. XP uses an iterative and incremental software process in relatively short cycles. Traditionally this type of approach would yield increased management overhead because management activities related to ending and starting iteration have to be executed for every iteration, but these are minimized in the XP process. XP introduces many practices but two techniques which are characteristic to XP are pair programming and test driven development [4]. XP may first seem quite chaotic, but it includes several good engineering practices [18]. However, for example, inadequacy of requirements management practices have raised some concerns [16].

Product development in the XP process starts with "planning game." Planning game can be divided into "release planning" and "iteration planning" [5]. During planning game, customer writes user stories, which the developers estimate and customer then subsequently prioritizes. Planning game is a phase in XP development when requirements, that is stories, are elicited, estimated and selected for release. Planning game is performed for each release. A release is divided into iterations. The subset of stories based on priority and size is then selected for each iteration. This is called iteration planning. Developers then divide stories into tasks and give an estimate for each task. Estimating user stories is difficult in other than very coarse level. On the other hand, estimating tasks is much more easy and accurate because tasks are defined in more detailed and concrete level. The next step in the XP process is development when the iterations are produced and released. Then acceptance tests are used to validate the completion of stories. Our consideration in this paper focuses on planning game phase of the XP process.

2.2 Requirements Management (RM)

Requirements management (RM) can be seen as a parallel support process for other requirements engineering processes [20][12]. It ensures that requirements are documented and traceable during product development and changes to them are properly handled.

Requirements identification is an essential pre-requisite for RM. It focuses on the assignment of unique identifier for each requirement [20]. These identifications can be used to unambiguously refer to requirements during product development and management. Further, requirement attributes can be used to record additional information about requirements [13]. Leffingwell and Widrig [13] emphasize that by using attributes, you get better management of complexity of information.

Requirements traceability (RT) refers to the ability to describe and follow the life of a requirement in both a forwards and backwards direction [9][10]. Gotel [9] emphasizes the life cycle aspect of the traceability. Requirements form the basis for design and implementation activities, and they should be traceable through product's life. Requirements' traceability is needed, e.g. for verification and change impact analysis activities. Traditional solutions for RT are based on manual traceability tables (e.g. solutions based on spreadsheets) and automatic management of traceability information stored in database management system (e.g. RM tools).

Requirements change management refers to the ability to manage changes to the requirements [12]. It also ensures that similar information is collected for each proposed change and that overall judgments are made about the costs and benefits of proposed change. Requirements are "frozen" when moving to the design phase (requirements baselining). Even if requirement specification is comprehensive, something can change during development, for example, customer's needs or regulations. This causes the need for clear practices that guide how possible changes to requirements are handled.

2.3 Related work

Currently, XP has generated a lot of interest from practitioners and academia. However, the management of user stories has received only little attention in the respective XP literature. A user story can be thought as a high level requirement or a user requirement. It is the user's conception about a functionality that the system should provide. RM and configuration management (CM) are only implicitly addressed by XP. It may be that the method developers viewed RM and CM too bureaucratic, heavy weight or ceremonial [7] to be included in XP. Currently, in the XP process, user stories and tasks are written on paper cards [4]. Story and task cards are identified using story and task numbers, and story cards are placed onto the wall with their respective task cards. This procedure was used in the eXpert -project, an empirical XP case study carried out in VTT Electronics [1]. This is a very simple and powerful way of visualizing the traceability between stories and tasks when a project team works in co-located workspace.

Few authors have considered requirements engineering in XP development. Paetsch et al. [17] analyze the commonalities and differences of traditional RE and agile SW development. They further analyze possibilities how agile methods can benefit from traditional RE methods. Breitman and Leite [6] support XP by using a scenario structure to organize information elicited through the user stories. They do not agree with Beck that implemented stories should be discarded but highlight the traceability of stories. Alike, Wagner [21] concludes that the lack of written, traceable requirements can make it difficult to maintain the software over time. He also emphasizes that user stories are not complete requirements, but requirements are actually spread into stories, test cases and code. On the other hand, Wagner [21] states that requirements baselining exists, in some form, in the XP process, because each iteration contains agreed set of stories. Internet-based tool support for distributed XP, called MILOS, has been introduced by Maurer and Martel [15]. Solution supports virtual software teams with communication, collaboration and coordination. The solution allows you to write and manage stories and tasks in electronic format. Lippert et al. [14] claim that a computer cannot be used for the planning game. On the other hand, they further specify that a computer can be used just for writing stories and tasks and printing them out on paper. The XP process itself does not exclude the use of automated tools for storing user stories. However, it is obvious that they still should be printed and put onto the wall if necessary.

Pinheiro [19] discusses requirements research from the point of view of agile methods, especially from XP viewpoint. Paulk [18] considers XP from software CMM (Capability Maturity Model) perspective and concludes that XP has many good practices, but they are not suitable for every occasion. Requirements process modifications in the XP process are introduced by Nawrocki et al. [16]. They assess XP against CMM-model and model introduced by Sommerville and Sawyer [20]. They state that the main weaknesses of the XP approach

to requirements management is the lack of requirements documentation. This causes problems especially when managing changes to requirements and maintaining traceability. According to the model introduced by Sommerville and Sawyer, XP supports only few practices. Thus, Nawrocki et al. [16] introduced an extended XP process that address the most of the deficiencies of the traditional XP process, but still remains lightweight. Grunbacher and Hofer [11] complement XP with EasyWinWin requirements negotiation technique. They state that this approach has the following benefits: emphasis of shared vision, more complete stakeholder identification, full perspective for on-line customer and extensive stakeholder involvement in decision-making.

The literature identifies the storage and documentation of requirements as one problem area in XP. This activity can be included and partly automated in XP, in some form, but the challenge is not to jeopardize XP's intentions to remain agile.

3 Principal findings from an empirical case study

This section introduces our findings based on the analysis of empirical case study. For details of this case study, see [1]. According to our study and experiences, the following issues were recognized especially important when considering the documentation and storage of user stories and tasks in XP.

First, XP provides the basic set of practices that should be adapted into the development situation. This means that practices are tailored for the purposes of a project and the organization, and they can be further changed on-the-fly on periodic process assessments, i.e. if they do not work. For example, some attributes in story and task cards were found unnecessary during the experiment. Thus, the tool support should not force detailed procedures but provide just enough basic abilities for defining project-specific user story and task attributes as well as abilities for storing, relating and retrieving user stories and tasks.

Second, the tool should allow the XP process to remain agile. This means that the tool should not jeopardize XP's intentions for open communication and lightweight management and documentation. Paper cards were used in experiment for story and task documentation. Story cards were placed onto the wall with their respective task cards. This was an efficient way of visualizing the dependencies between stories and tasks when operating co-located workspace. In XP methodology, a customer is always present in the project room and communicates with the project team. However, this is not always possible in real life. Thus electronic storage, management and distribution of story and task cards were considered important if the customer cannot always be present.

Third, requirements management tool support should be integrated into the project's overall development environment. This allows the project team to operate via one channel from a user story definition, through implementation, up to testing. The integration should also store and manage all project-related information under the appropriate project. This means that the project personnel can easily find all relevant project-related data from the system.

4 Integrated tool support for the management of user stories

In this section, we introduce our solution for the management of user story and task cards called StoryManager. In specific, the proposed solution has been integrated in Eclipse –environment (Eclipse project [8]). Eclipse is a development environment and a tool integration framework found suitable for XP development. Eclipse –environment was used successfully in VTT's XP case study [1]. Tools are integrated through plug-ins into Eclipse. Currently there are hundreds of plug-ins available. In VTT's experiment, the Eclipse environment contained integrations with CVS (Concurrent Versions System), JDT (Java Development Tooling) and Junit (Testing framework). This framework complemented with StoryManager extension enables integrated tool environment for XP-based development from stories through implementation up to testing. Eclipse allows the user to easily navigate between tool perspectives during product development (Figure 1).

Our intention has been to remain agile since any new solution or practice should not jeopardize the fundamental idea of adaptable and lightweight processes. We maintain that the solution proposed is flexible enough to be tailored for an individual XP project with the respects of a project's needs. On the other hand, it integrates the basic functionality needed for adequate user story and task definition and management in the integrated development tool framework (Figure 2). Integrated tool framework enables the project team to work with one channel throughout the whole development life cycle. During planning game, the team works through StoryManager plug-in. Stories and tasks are stored into MySQL relational database. During implementation and

testing, the team works, e.g. through JDT plug-in and Junit view. From information management point of view, our approach includes support for requirements management and configuration management. Requirements management (StoryManager) is used in this environment to share and manage user stories and tasks. On the other hand, configuration management (CVS) is used to manage and share code and other documentation.

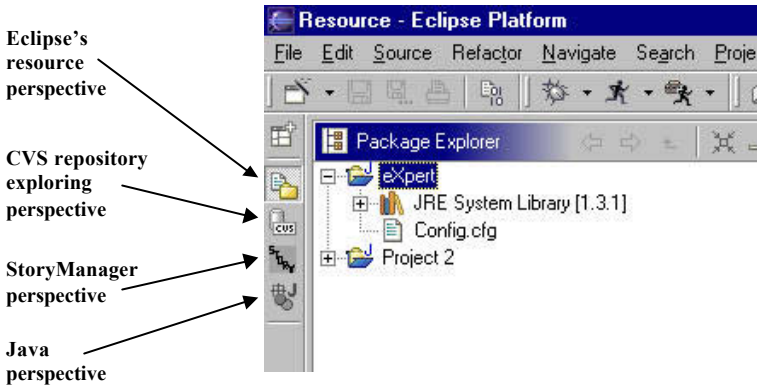


Figure 1: Navigation between perspectives in Eclipse

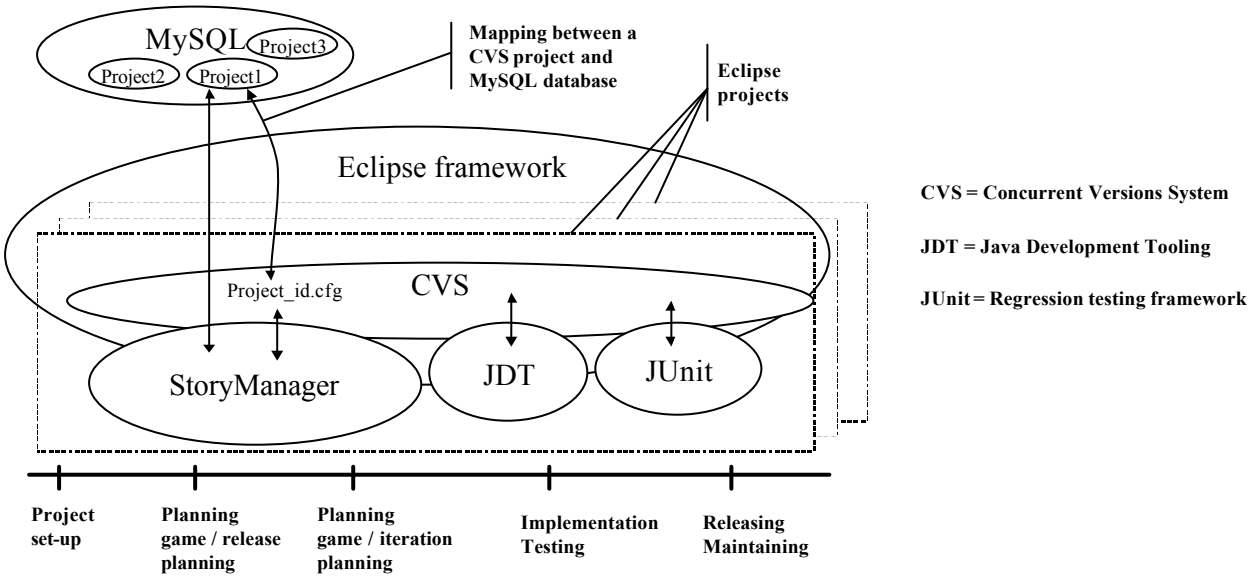


Figure 2: StoryManager as part of integrated tool environment for XP development

The introduction of StoryManager facilities is divided into process steps. Each process step introduces relevant application facilities and explains their usage. Phase "Project set-up" is used to describe activities that need to be performed before planning game:

Project set-up

First, the relevant Eclipse project is selected or created. Attribute definition facility is used to define user-defined attributes for stories and tasks. This facility allows flexible story/task attribute definitions according to the needs of the project (Figure 3). Then StoryManager creates project-specific database under the Eclipse –project. The program creates project-specific configuration file (CFG-file) which will be stored in CVS into project-root (Figure 2). The CFG-file indicates for StoryManager the correspondent MySQL database with the respect of the selected CVS project.

Planning game / release planning

First during the Planning Game -phase, a customer defines stories in cooperation with a project team. The program enables to fill-in story/task cards according to project-specific attributes. The program's AutoID facility is used to create automatically unique identifiers (ID) for stories (and tasks). This facility corresponds with requirements identification activity. Certain attributes are mandatory including:

- ID: automatically created by the system (story_xxx, task_xxx)
- Status: Defined, Implementing, Done, Postponed (color codes are used in tree-view to indicate status)
- Release: Not specified, 1, 2, 3, …
- Iteration: Not specified, 1, 2, 3, …
- Description: actual text for story / task

Other attributes are user-defined and optional. The program allows the user to modify stories, but in XP, only the last story version is relevant. Thus, the application contains only the last updated version. According to basic XP principles, formal and bureaucratic change management activities are not appropriate. However, the application stores a version history from the item (story/task) which can be used to examine the history of the item (Figure 3). Release and Iteration -attributes illustrate the selection of items for certain release and for certain iteration. Actually this corresponds with requirements baselining facility indicating agreed set of items for certain release and iteration.

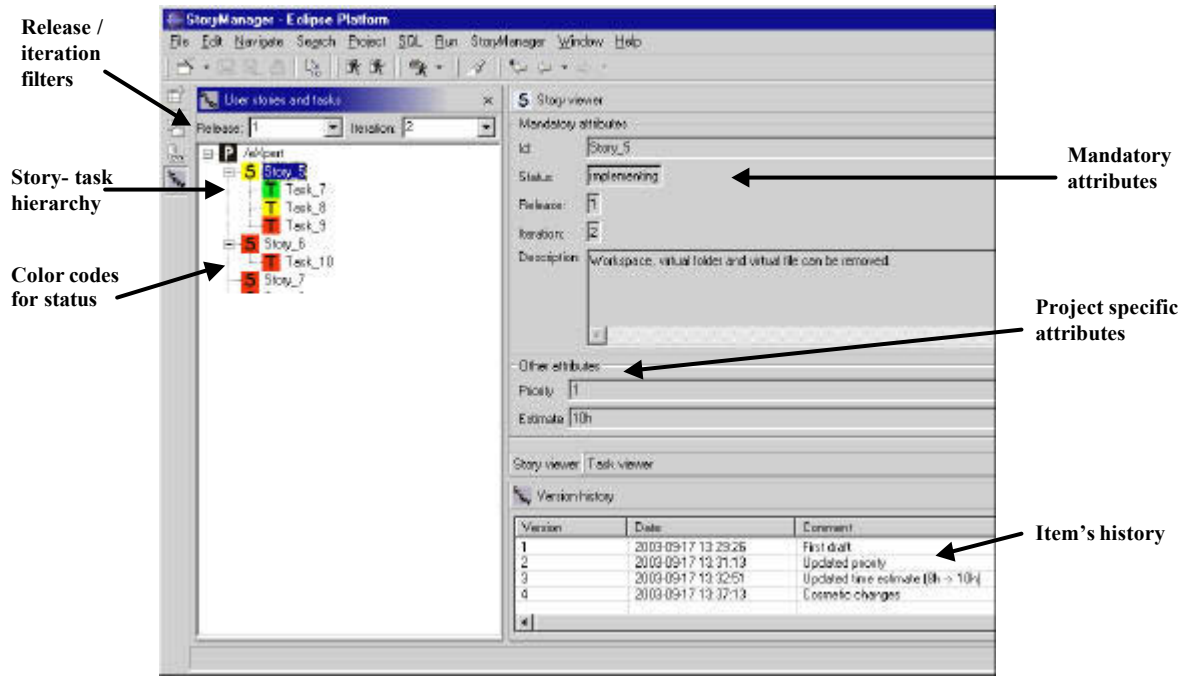


Figure 3: StoryManager main perspective.

Planning game / iteration planning

During iteration planning, the certain set of stories is selected for next iteration. This will be illustrated using Iteration -attribute. Then tasks are defined for next iteration. Mandatory attributes and main facilities are same for story and task cards, but optional attributes can vary. In addition to this, the task can be related under a story (traceability between stories and tasks) (Figure 3). In this case, the program stores the linking information in MySQL Link-table. However, a task can be created also under a project root and related afterwards.

Process phase independent facilities

Certain facilities are needed despite of a process phase. These facilities include the check out/in, views and reporting. Parallel story/task modification has been disabled to avoid uncontrolled changes when someone is modifying a story or task. When a user check out a story or task from database for modification, the system locks the story or task and it cannot be modified or deleted by others. When the user checks in the item, the system unlocks it and after that other users can modify it. Views are used to describe StoryManager database's content for a user (retrieving information) (Figure 3). The basic StoryManager perspective contains tree-view containing hierarchical story/task structure, story/task –content view and history view. Hierarchical story/task structure can be filtered by using release and iteration attributes. Finally, reporting facilities are used to create reports from the contents of database. This facility can be used if stories and tasks need to be printed and put onto the wall. Currently the following reports are possible: all stories, all tasks, stories and corresponding tasks, stories and tasks based on selected release and iteration (baseline).

5 Conclusions and future research

Requirements management and configuration management are only implicitly addressed by XP. Requirements management is needed to ensure that requirements are identified, traceable and all changes to requirements are properly handled. However, in a dynamic and fast moving XP project, traditional RM may tie up too much resources. In this paper, we introduce an approach for supporting requirements engineering in XP by managing user stories and tasks, called StoryManager. Traditionally this has been done manually using story and task cards. Our aim was to consider requirements management in XP and provide integrated, yet lightweight, approach for user story and task management. Integration was carried out in Eclipse –environment. The advantages of solution are the following.

First, the approach integrates all information from requirements (stories) through implementation up to testing under the appropriate Eclipse –project. Users can operate through one environment that store and control project-related data in electronic format including stories, tasks, code and other documentation.

Second, StoryManager allows you to define project-specific attributes for stories and tasks. This is a very simple way to define templates to collect minimum information about stories and tasks with the respect of each project.

Further development of solution includes the validation of the first version of the solution. Further research should also consider integration between stories and implementation/testing. The validation will analyze the applicability of solution for XP and verify if the solution remains lightweight. Bureaucratic and complicated solution just jeopardize simplicity. Empirical validation of the proposed system is currently ongoing. The verification is carried out in the XP experiment project developing mobile application software in Eclipse environment. The experiment project works according to XP practices, but the customer is off-site. Thus, the customer can follow the implementation of stories from the remote office using StoryManager.

The limitation of our solution is that the StoryManager is not a stand-alone program, but it has to be used in the context of Eclipse environment. Therefore, it dictates the development environment. Also the reporting facilities and visualization of user stories and tasks is challenging.

Acknowledgments

The work was funded by the Technical Research Center of Finland (VTT) and by the National Technology Agency (TEKES). This work has been part of the ITEA project called MOOSE (http://www.mooseproject.org) and VTT's strategic research project called AGILE (http://agile.vtt.fi).

References:

[1] Abrahamsson, P.: Extreme programming: First results from a controlled case study. Euromicro 2003, Antalya, Turkey, 2003.

[2] Abrahamsson, P., Salo, O., Ronkainen, J., Warsta, J.: Agile software development methods: Review and analysis, VTT Publication 478, Espoo, 2002.

[3] Abrahamsson, P., Warsta, J., Siponen, M., Ronkainen, J.: New directions on agile methods: A comparative analysis. International Conference on Software Engineering (ICSE25), Portland, Oregon, USA, 2003.

[4] Beck, K.: Extreme Programming Explained: Embrace Change. Reading, Massachusetts: Addison-Wesley, 1999.

[5] Beck, K., Fowler, M.: Planning extreme programming. Addison-Wesley, 2000.

[6] Breitman, K., Leite, J.: Managing User Stories. International Workshop on Time-Constrained Requirements Engineering, TCRE 02, 2002.

[7] Cockburn, A.: Agile Software Development, Boston, Addison-Wesley, 2002.

[8] Eclipse project.: http://www.eclipse.org/. Available in 29th September 2003.

[9] Gotel, O.: Contribution Structures for Requirements Traceability, Ph.D. Thesis, Imperial College of Science, Technology and Medicine, University of London, August, 1995.

[10] Gotel, O., Finkelstein, A.: An Analysis of the Requirements Traceability Problem, Proceedings of the First International Conference on Requirements Engineering, pp.94-101, 1994.

[11] Grunbacher, P., Hofer, C.: Complementing XP with Requirements Negotiation. XP2002.

[12] Kotonya, G., Sommerville, I.: Requirements Engineering: Process and Techniques, John Wiley & Sons, 1998.

[13] Leffingwell, D., Widrig, D.: Managing Software Requirements - A Unified Approach, Addison-Wesley, 2000.

[14] Lippert, M., Roock, S., Wolf, H.: eXtreme Programming in Action. John Wiley & Sons, 2002.

[15] Maurer, F., Martel, S.: Process Support for Distributed Extreme Programming Teams, ICSE 2002 Workshop on Global Software Development, 2002.

[16] Nawrocki, J., Jasinski, M., Walter, B., Wojciechowski, A.: Extreme Programming Modified: Embrace Requirements Engineering Practices, 10th IEEE Joint International Requirements Engineering Conference, RE'02 Essen, Germany, September 2002.

[17] Paetsch, F., Eberlein, A., Maurer, F.: Requirements engineering and agile software development. IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE-2003 / KMDAP2003), June 9-11, Austria, 2003.

[18] Paulk, N.: Extreme programming from a CMM perspective, IEEE Software , Volume: 18 Issue: 6, Page(s): 19 -26, Nov/Dec 2001.

[19] Pinheiro, F.: Requirements Honesty. International Workshop on Time-Constrained Requirements Engineering, TCRE 02, 2002.

[20] Sommerville, I., Sawyer, P.: Requirements Engineering: A Good Practise Guide. John Wiley & Sons, 1997.

[21] Wagner, L. : Extreme Requirements Engineering. Cutter IT Journal, Vol. 14, No. 12, December 2001.

PAPER IV

Improving requirements management in extreme programming with tool support

An improvement attempt that failed

30th Euromicro Conference, EUROMICRO 2004, Rennes, 31 Aug.–3 Sept. 2004. IEEE Computer Society. Pp. 342–351.

Improving Requirements Management in Extreme Programming with Tool Support – an Improvement Attempt that Failed

Jukka Kääriäinen, Juha Koskela, Pekka Abrahamsson, Juha Takalo
VTT Technical Research Centre of Finland
P.O. Box 1100, FIN-90571 Oulu, Finland
{jukka.kaariainen; juha.koskela; pekka.abrahamsson; juha.takalo}@vtt.fi

Abstract

While Extreme programming (XP) relies on certain principles, it requires an extensive set of tools to enable an effective execution of its practices. In many companies, putting stories on the board may not be sufficient for managing rapidly changing requirements. The objective of this paper is to report the results from a study where a requirement management tool – the Storymanager – was developed to meet the needs of a XP project team. The tool was used in a case project where a mobile application for real markets was produced. The tool was dropped by the team only after two releases. The reasons of the process improvement failure are addressed in this paper. The principal results show that the tool was found to be too difficult to use and that it failed to provide as powerful a visual view as the paper-pen board method. The implications of these findings are addressed for both the practitioners and researchers in the field.

1. Introduction

Extreme programming (XP) is a well known agile software development method. While XP relies on certain principles, such as communication and simplicity, it also requires tools to enable an effective execution of its practices. In many companies, listing the stories on to the board is not sufficient for managing changing requirements. We made an attempt at finding a solution for managing user stories and tasks in electronic format as a part of the Eclipse tool integration framework, but there were no solutions available for this. Thus, we decided to develop a specific plug-in application for the Eclipse environment, which was called the Storymanager. One of the characteristic of methods and tools is that they need to be adapted to fit a certain company or project context [1]. Thus, these tools have to take into account the nature of the project, including the development methods used in the project.

The objective of this paper is to report the results from a study where a requirement management tool – the Storymanager - was developed to meet the needs of a fast moving XP project team. The aim of the tool was to minimize rework and automate the time consuming paper-pen practices, such as recording story and task items on the board and then separately on an excel sheet, or equivalent.

The paper is composed as follows. The background concepts of Extreme Programming and Requirements Management (RM) are first introduced. Then the related research is laid out regarding RM in XP context. This is followed by a detailed discussion on our solution for RM in XP environment. Then, research design is described. Finally, the results are presented and discussed. The paper is concluded with final remarks and the identification of future research needs.

2. Background

This section introduces the concepts of extreme programming and requirements management.

2.1. Extreme Programming

Extreme programming (XP) as a concept has emerged in the late 90's [2]. Along with XP, several agile methods have emerged (for an overview, see, e.g., [3]). XP addresses the issues of changing requirements and their cost by simplifying management tasks and documentation. XP uses an iterative and incremental software process performed in relatively short cycles.

Product development in the XP process starts with a "planning game." Planning game can be divided into "release planning" and "iteration planning" [4]. During

Proceedings of the 30th EUROMICRO Conference (EUROMICRO'04)
1089-6503/04 $ 20.00 IEEE

the planning game, the customer writes user stories, which are estimated by the developers and then prioritized by the customer. After this, developers divide the stories into tasks and give an estimate for each task. The next step in the XP process is the actual development, during which the iterations are produced and released. Then finally, acceptance tests are used to validate the completion of stories.

2.2. Requirements Management

Requirements management (RM) can be seen as a parallel support process for other requirements engineering processes [5, 6]. It ensures that requirements are documented and that they are traceable during product development and that changes to them are properly handled.

Requirements identification is an essential pre-requisite for RM. It focuses on the assignment of an unique identifier for each requirement [5]. These identifications can be used to unambiguously refer to requirements during product development and management. Further, requirement attributes can be used for recording additional information about requirements [7].

Requirements traceability (RT) refers to the ability to describe and follow the life of a requirement in both forward and backward direction [8, 9]. Gotel [8] emphasizes the life cycle aspect of traceability. Requirements form the basis of design and implementation activities, and they should be traceable through the life-cycle of a product. Requirement traceability is needed, e.g., for verification and change impact analysis activities.

Requirements change management refers to the ability to manage changes to requirements [6]. It also ensures that similar information is collected for each proposed change and that overall judgments are made about the costs and benefits of a proposed change. Even if requirement specification is comprehensive, some changes can take place during development. This gives rise to the need for clearly defined practices that provide guidance for handling possible changes to requirements.

3. Related research

In this section, the related research is presented. The results of this review are used for building research lenses (an analysis framework), which will be used to analyze the results of this case project later in the paper.

Traditional XP relies on efficient communication, which is one of the basic values of the method [2, 10]. XP emphasizes communication, e.g. through practices, such as “Open Workspace”, "Pair Programming" and “Planning Game” [2, 11]. Macias et al. [12] state that interactive communication between the developers, clients and managers in XP should be emphasized. Face-to-face communication is an efficient mechanism in realizing this. For an agile team to be successful, good communication mechanisms have been found to be critical [13].

The agile principles value working software over comprehensive documentation [3]. Beck [2] also emphasizes lightweight documentation in XP based development. Ambler [13] emphasizes the slogan “Travel light” in the context of documentation. Ambler states that it is useful to produce just enough documentation and to update it only when needed. This enables the team to be more effective in producing results that deliver more business value for the customer than traditional paper-driven methodologies.

Ease-of-use is an important aspect when developing tool support for XP development (and actually for any SW related work). For example, Lippert [14] identifies ease-of-use as a very important aspect for XP tool support during continuous integration. The tool should not slow down the product development or cause additional maneuvers during fast-paced development. O'Brian Holt [15] present some factors for assessing usability, including aspects such as: Is the system easy to learn to use? Is it possible to modify the system without reducing its usability? Is the system comfortable and satisfying to use? Nielsen [16] defines the different aspects of usability as follows: easy to learn, efficient to use, easy to remember, few errors, and subjectively pleasing.

Some authors have considered requirements management from an XP point of view. Breitman and Leite [17] support XP by using a scenario structure to organize information elicited through user stories. While they do not agree with Beck who maintains that implemented stories should be discarded, they highlight the traceability of stories. Nawrocki et al. [18] state that the main weaknesses of the XP approach to requirements management is the lack of requirements documentation. This causes problems especially when managing changes to requirements and maintaining traceability. Alike, Wagner [19] concludes that the lack of written, traceable requirements can make it difficult to maintain the developed software over time. On the other hand, Wagner [19] states that requirements baselining exists, in some form, in the XP process, because each iteration contains an agreed set of stories.

From a change management viewpoint, the requirements management literature in fact proposes quite rigorous processes for managing requirement changes [20]. However, formal and cumbersome practices for change management do not fit the nature of XP. Therefore, lightweight and simple practices for managing changes in XP have turned out effective in practice [21].

Several authors have addressed the tool support used for managing user stories and tasks. Internet-based tool support for distributed XP, called MILOS, has been introduced by Maurer and Martel [22]. This solution supports virtual software teams with communication, collaboration and coordination. The solution allows the user to write and manage stories and tasks in electronic format. Rees [23] has presented a tool called DotStories for managing user stories, claiming that the tool approaches an ideal solution for user story management. Rees also refers to spreadsheets and databases as further potential tools for managing user stories. Lippert et al. [24] claim that a computer cannot be used for the planning game. On the other hand, they further argue that a computer can be used just for writing stories and tasks and printing them out on paper. The XP process itself does not exclude the use of automated tools for storing user stories. Actually, tools and databases can provide a means for more effective information management [25] [5].

Integrated environment and data sharing enable the project team to focus on development work, while daily data management has been automated. This means that all project-related data is managed at a unified location and integrated tool support eases tedious tasks, such as information retrieval, distribution, consistency checking, archiving, etc. It has been stated in literature that management system integration is likely to improve the consistency and sharing of product-related information (e.g. [26] and [27]).

4. Tool support for the management of user stories: the Storymanager tool

Our solution to managing user stories and tasks is called Storymanager. The proposed solution was integrated in the Eclipse environment. Eclipse is a development environment and a tool integration framework found suitable for XP development. A detailed description of the proposed Storymanager solution has been published in [28] (Figure 1).

Our intention was to remain agile, no new solution or practice should jeopardize the fundamental idea of adaptable and lightweight processes. The basic intention of this study was to transfer manual XP requirements management practices into an electronic form and yet try to remain agile. A further aim was to integrate the solution into the Eclipse tool integration framework. An integrated tool framework enabled the project team to work with one channel throughout the whole development life cycle. During the planning game, the team was working through the Storymanager plug-in. Stories and tasks were stored into an MySQL relational database. During implementation and testing, the team was working, e.g., through a JDT (java development environment) plug-in and a Junit (testing framework) view. From an information management point of view, our approach included support for requirements management and configuration management. Requirements management was used in this environment for sharing and managing user stories and tasks, while configuration management (CVS) was used for managing and sharing code and other documentation.

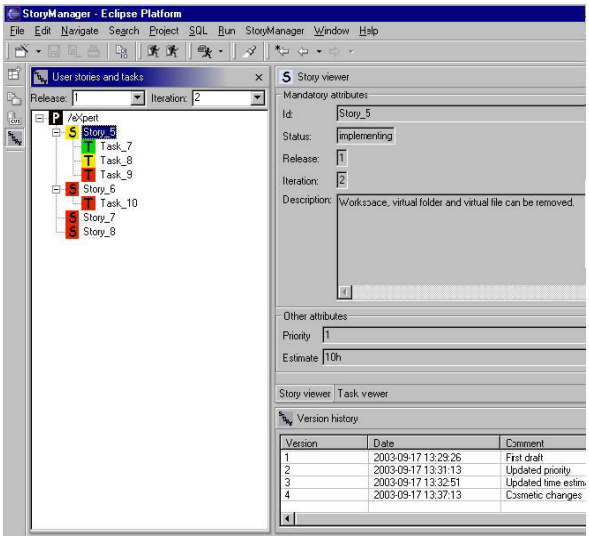


Figure 1. Storymanager - the main view

The Storymanager allows a specification of story and task attributes according to the needs of the project. The program enables filling in story/task cards according to project-specific attributes. The AutoID facility of the program is used to automatically create unique identifiers (ID) for stories (and tasks). Certain attributes, however, are mandatory, e.g. status, description, release identifier, iteration identifier. Other attributes are user-defined and optional.

The program allows the user to modify stories, but in XP only the last story version is relevant. Thus, the application contains only the last updated version.

According to basic XP principles, formal and bureaucratic change management activities are not considered appropriate. However, the application stores a version history of the item (story/task), which can be used for examining the history of the item. The attributes "Release" and "Iteration" contain information about the selection of items for specific releases and iterations. In fact, this corresponds with the requirements baselining facility indicating an agreed set of items for a specific release and iteration.

During iteration planning, a set of stories is selected for next iteration. This is illustrated using an iteration attribute. Then the tasks are defined for next iteration. A task can be assigned to a story (traceability between stories and tasks). In this case, the program stores the linking information in a MySQL Link-table. However, a task can also be created under a project root and allocated afterwards.

Certain supporting features are needed regardless of the phase of the project. These features include check out/in capabilities, views and reporting. The reporting features are used for printing stories and their respective tasks and putting them on the board when operating in an open workspace.

5. Research design

This section describes how the research and experiment settings were designed.

5.1. Research settings

In this chapter, the research settings used for developing and validating the solution designed for managing user stories are depicted. Application development and validation are based on two XP experiment projects (Figure 2) using the action research [29, 30] approach as the principal methodological driver. Avison [31] and Fowler & Swatman [32] have used the action research method to build information system development methods. Action research is done in cycles, each cycle consisting of planning, action, observation and reflection phases. After each cycle, there will be a revised plan for the next cycle as a result. We applied the action research approach while trying to improve the management of user stories and tasks in XP development.

A project called eXpert was used for developing a system for managing the research data and documents at VTT. The project used XP practices for developing the system. Detailed results of the eXpert project can be found in [33]. The project used a manual solution for managing user stories and tasks, as suggested in the XP literature. During the project, the need for electronic user story and task management emerged. After the project, the results were analyzed and an application was constructed to support a more automated, i.e. electronic, user story management. The requirements for electronic story management and the application itself were introduced in [28]. The validation and improvement planning of the application were carried out as part of the zOmbie-project, which was concerned with developing mobile application software in the Eclipse environment. The validation of the electronic user story management tool was carried out and observations and interviews were made during the project. After the project, the results were analyzed and improvement ideas were produced for future development.

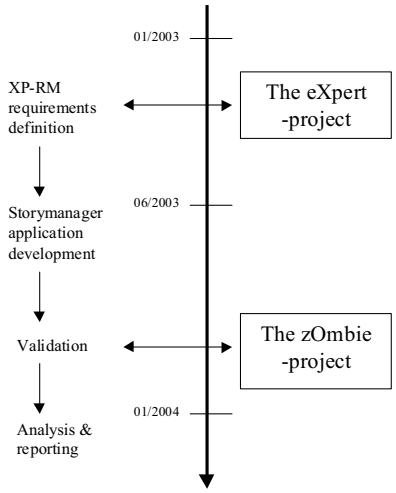


Figure 2. Storymanager development

An analysis framework was constructed to analyze and understand the results of the validation. The framework was based on the survey of related research and underlying XP concepts and requirements management, as presented in sections 2 and 3 of this paper. Our analysis framework reflects technical issues as well as those concerning the methodological aspects of XP. The technical issues focus on the definition of functionality that is needed for requirements management in an integrated XP development environment. The methodological aspects refer to the underlying nature of the XP method. Table 1 presents the analysis framework.

Table 1. Analysis framework

Perspective	Description	Key references
Communication	Does the solution allow open communication between developers and between developers and customers?	[2, 10, 11]
Documentation	Does the solution allow lightweight documentation?	[2], [3], [13]
Ease-of-use	Is the solution easy to use, so as to support fast-paced iterative development?	[14], [15], [16]
Functionality	Does the solution support functional needs for requirements management (identification, traceability and change management) and integrated development environment?	Requirements management : [5], [6], [7], [8], [9], [21], [17],[19], [24] Integrated development environment: [26], [27]

5.2. Experiment settings

The functionality of the application was validated and tested in an experimental XP project called zOmbie. In the zOmbie project, mobile application software was developed in the Eclipse environment. In this experiment, the Eclipse environment was complemented with Storymanager. The aim of the zOmbie project was to produce a real financial sector software product for real markets. The project was an engineering success. The product is now being marketed. The Eclipse environment was used successfully already in the previous XP case study of VTT [33]. In zOmbie case study, the Eclipse environment was used together with the following tools (Table 2).

Table 2. Tool environment in the zOmbie experiment

Tool	Version	Description
Eclipse	2.1	Tool integration framework
Storymanager	1.0.0	User story and task management
JDT (part of Eclipse package)	2.1	Java development environment
Junit	3.8.1	Testing framework
CVS	1.11.2	Version management

The verification was carried out in the XP experiment project developing mobile application software in Eclipse environment. The project team consisted of 5 developers and a project manager. The project worked according to XP practices, but some of the practices needed slight adaptation (e.g., test-first development in mobile application is challenging) according to the business needs.

The aim of Storymanager validation was to use the XP project to verify our solution for requirements management. The focus was to ensure that requirements management support was adequately considered in the integrated development environment and that the solution allowed the XP project to remain "agile" and "lightweight". The project team was allowed to systematically change any practices if they felt that these did not work. Thus the project group were trained and encouraged to "think according to XP values".

Quantitative and qualitative data were collected throughout the project. The project had nominated a person responsible for metrics, who was monitoring that data was collected systematically. The metrics and practices for gathering quantitative data were defined before the project start-up. Qualitative data was collected from the project team by using a specified comment template. The template contained questions about the applicability of the Storymanager solution. The comments were processed and analyzed using Post Mortem [34] sessions. The role of XP coach was extended in zOmbie. The coach was also acting as a Storymanager advisor, collecting experiences concerning its usage. Even after the project, comments and improvements were inquired from the project team and coach. The inquiry performed after the project contained the following questions, which were formulated based on the experience that the

Storymanager was abandoned after two releases and manual story and task management was used during the rest of the project:

- Which were the advantages of Storymanager compared to manual story and task management?
- Which were the disadvantages of Storymanager compared to manual story and task management and why was the Storymanager abandoned?
- Give other comments and suggestions for the improvement of electronic management of user stories and tasks?

6. Results

Table 3 presents the basic quantitative data from the experiment. It provides basic information about the size and schedule of the project and helps the reader to understand the nature of the project where the Storymanager tool was used.

Table 3. Background information about the mobile application case project

Collected data	Release 1	Release 2	Release 3	Release 4	Release 5	Correction release	Total
Calendar time (weeks)	1	2	2	2	1	0.4	8.4
Total work effort (h)	115.3	238.9	273.2	255.6	123.7	66.8	1073.5
Planning day effort (h)	37.1	22.7	32.8	24.5	15.7	13.5	146.3
# User stories implemented	3	4	5	5	1	1	19
# Tasks implemented	11	25	18	18	10	10	92

The development team used Storymanager for storing, distributing and retrieving story and task information during the first two releases. Stories and tasks were created, modified and maintained in Storymanager and printed out from the system and put on the board. After the second release, the project team moved into manual story and task management, because of the visualization and usability problems of electronic story and task management. The project manager had some experience with manual management of user stories and tasks, which made it possible to move from electronic to manual management.

This section introduces the results of the interviews. First, the project team and coach were asked to voice their opinion about the advantages of the Storymanager for story and task management. A collection of answers from the zOmbie coach and team members are presented in the following:

"Easy to operate with stories and tasks when they are in electronic format. In manual format, story or task modification required rewriting the whole card ."
"Integration to Eclipse enables easy access to tool."
"Manual cards are sometimes lost, but when they are in electronic format, they are easily accessible."
"The use of the status attribute was easy and the color codes were practical."
"The customer can easily follow the implementation of stories from a remote office using Storymanager"
"Manual archiving is not needed after a project"

Then the project team and coach were asked about the disadvantages of Storymanager for story and task management and they were also asked to give the reasons for abandoning the tool. The answers received from the zOmbie team members and coach are listed in the following:

"Not clear. It is easier to see the Big Picture of the project (e.g. status) when the manual story and task cards are put on the board"
"The reports (layout of printed story/task cards) were not good. If they were better, it would be more useful."
"The tasks in Storymanager are in text format. However, tasks sometimes also contain other formats than just pure text (i.e. pictures, etc.)."
"It was difficult to move tasks between stories."
"The usability was not good."
"The AutoID functionality was confusing in Storymanager."
"More disadvantages than advantages."

The project team and coach were also asked to give further comments and suggestions for improvement. In the following, a summary of answers received from zOmbie team members and coach is presented:

"Support is needed for XP project management."
"A tool should be easy to use."
"It should be easy to see the Big Picture of the project when using the tool."
"The tool should allow moving tasks more easily between stories."
"The tool should allow linking tasks with application code."
"Printing of stories and tasks with bigger font."

Table 4 summarizes the advantages and disadvantages of the Storymanager. The reasons for moving from electronic management to manual management and their implications are analyzed in next section.

Table 4. Summary of the results gained from the Storymanager tool study

Perspective	**Results** ***("+" strengths," –" weaknesses/enhancements)***
Communication	+ The customer can easily follow the implementation of stories from a remote office using Storymanager + Color codes can be used to visualize the status of the story/task on computer screen – the Storymanager system or printed story/task reports do not make it easy to see the overall status ("big picture") of a project
Documentation	+ Manual stories or task cards need not be archived separately – Tasks can contain just text description – Reports are not clear (layout and font size)
Ease-to-use	+ It is easy to manage stories and tasks (modification) + Stories and tasks are easily accessible in electronic format – General usability of the tool is not good – Moving tasks between stories is difficult
Functionality	+ Integration to Eclipse enables easy access to tool and information + Tool provides reliable means to store, modify and retrieve information – AutoID provides "meaningless" code for a story or task – Support for XP project management should be added to the tool – There should be a possibility of linking tasks with application code

7. Discussion

This section analyses the results against the analysis framework defined in section 5.1. Agile methods, such as XP, aim towards efficient communication and lightweight documentation. Although some additional or modified XP practices were used in the VTT zOmbie-project, the basic development philosophy relied on open communication and lightweight documentation. As presented in related literature, [e.g. 5, 25, 35, 36], the adaptation of product information management should be made on the basis of the business context of a project. Thus, in this case, the nature of the XP method drives the adaptation of requirements management tool support. Table 5 summarizes the most important findings and their implications.

When considering the results in section 6, it can be noted that the developed solution for electronic management of user stories and tasks tackles mainly the same things as the manual one. The clear advantages of the electronic solution compared with manual management are related to the ability to reliably store, modify, distribute, retrieve and archive stories and tasks and the ability to operate in an integrated development environment where all development tools are available.

There are two fundamental differences between electronic and manual management. These are related to information visibility and usability. Open workspace allows the project team to use paper cards for stories

and tasks and put them on the board. This allows the team to get an overview of stories and tasks just by having a look at the board, discussing the items and making modifications directly to the story and task cards. While this way of working is, in fact, highly effective and it emphasizes natural interaction between developers, it does require that the team members share an open workspace. Rees [23] states that one problem connected with using databases for managing user stories is that they provide just poor group visualization of all cards. Our experiment supports this claim. Of course, electronic management also allowed us to print the stories and tasks and then to put them on the board. This has been suggested by Lippert et al. [24], who first claim that a computer cannot be used for the planning game, and then further specify that a computer can be used just for writing stories and tasks and printing them out on paper. However, our experiment shows that not even this approach works, if the editing and maintaining of printed story and task cards takes up too much effort in fast-paced development. Furthermore, the format of printed cards should be highly distinct to be able to compete with manual ones. Thus, further research is needed to explore the possibilities for improving visualization in electronic management of user stories and tasks.

Table 5. Findings and their implications

Perspective	Findings	Implications
Communication	Electronic management of stories and tasks enables remote customers and other stakeholders to view the status of the project in real-time. Electronic management seems to be an obvious solution when operating in a distributed development environment but it can jeopardize natural interaction between developers and the visibility of information in open workspace.	Visualization of stories and tasks is a challenge and requires further research. Furthermore, electronic management of user stories and tasks seems to be effective if the customer is off-site and the project is distributed over several sites.
Documentation	Electronic task cards should be able to contain also other formats of information than text descriptions. The format of reports should be clear.	Possibilities of integrating a drawing or modeling tool into Storymanager and Eclipse should be considered.
Ease-of-use	Tool usability should be good in fast-paced development.	Application development using, e.g., a User-Centered Design approach to ensure that usability issues are considered.
Functionality	The tool should also provide additional value for daily routines, not just automated support for the old “pen and paper” practices.	Support for XP project management and linking between tasks and application code should be examined as a part of the Eclipse environment.

It also became apparent during the zOmbie-project that electronic management should also allow other formats of information than just pure text (e.g. pictures, models, etc.). During the project, the operation with models was an important part of the work because the product being developed was complicated. On the other hand, in the eXpert-project, graphical modeling was not an important aspect. Therefore, the product being developed itself seems to affect the requirements management needs of the project. This claim is supported, e.g., by Kotonya and Sommerville stating that the type of system under development affects the requirements management solution [6]. The use of modeling in XP has been considered by Beck [2], who argues that although modeling can be used during the XP process, the pictures should not be saved. We cannot, however, agree with this proposition. If models are made, they should be archived as any other information during the project for maintenance reasons. On the other hand, modeling support easily makes the Storymanager tool complex.

If a project is distributed over several sites, electronic management of stories and tasks seems quite an obvious solution. This has been demonstrated by Maurer and Martel [22], for example. However, one

problem encountered with Storymanager had to do with its usability. In fast-paced development, the usability should be good. It is difficult to justify tool usage if the tool slows down the development and requires additional maneuvers. One way of tackling this kind of problem is to use User-Centered Design approaches during tool development, so as to ensure that usability issues are considered [37].

The basic problem in the Storymanager requirements management solution is that even though it does address the requirements management issues of product development, it also fights against the values of the XP method. The solution reduces simplicity and open communication between team members when operating in an open workspace during fast-paced product development.

The developed automatic solution focused mainly on the automation of manual story and task management activities. However, Tolvanen [1] states that, in the long run, the promise of tools does not lie just in the automated support of old "pen and paper" methods. Our findings support this claim. Consequently, the aim of supporting manual operations without providing extensive features or significant additional value for developers proved too narrow. Some of the comments gained from the zOmbie project team addressed this issue. For example, some enhancement ideas were voiced concerning providing support for project management using story and task efforts and related automatic calculation, and also concerning automated traceability between tasks and code.

8. Conclusions and future research

This paper presents the results from a study where a requirements management tool called Storymanager was developed to meet the needs of a fast moving XP project. The aim of the tool was to minimize rework and to automate time consuming paper-pen practices, such as recording story and task items on the board. The tool was used in a project where mobile application software for real markets was produced. The team abandoned the tool only after two releases. Essentially, the tool failed to provide as powerful a visual view as the paper-pen board method. The developed tool also turned out to be too difficult to use in fast-paced development environment. The interview data further revealed that a computerized solution is not by any means self-evident, but rather it has to be able to compete with the best alternate solutions, i.e. manual paper-pen approach in this case, and even provide additional value for the project team.

The experiences, findings and implications of this study should be of interest to any organization considering requirements management tool support for XP projects.

The results emphasize the role of adaptation in tool development. The tool, constructed to support any development method, should take into account the underlying values of the method itself. If this is not the case, the tool is likely to work against the nature of the method.

Future research will be concerned with constructing a new solution addressing the lessons learned that were found relevant during this study. This is to be followed by validating and enhancing the solution in future SW development projects.

References

[1] J. P. Tolvanen, "Incremental method engineering with modeling tools," in *PhD dissertation*. Jyväskylä University Printing House and ER-Paino Ky: University of Jyväskylä, Finland, 1998.

[2] K. Beck, *Extreme programming explained: Embrace change*. Reading, MA.: Addison Wesley Longman, Inc., 2000.

[3] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta, *Agile software development methods: Review and Analysis*. Espoo, Finland: Technical Research Centre of Finland, VTT Publications 478.

[4] K. Beck and M. Fowler, *Planning extreme programming*. New York: Addison-Wesley, 2001.

[5] I. Sommerville and P. Sawyer, *Requirements Engineering: A Good Practise Guide*: John Wiley & Sons, 1997.

[6] G. Kotonya, Sommerville, I., *Requirements Engineering: Process and Techniques*: John Wiley & Sons, 1998.

[7] D. Leffingwell and D. Widrig, *Managing Software Requirements - A Unified Approach*: Addison-Wesley, 2000.

[8] O. Gotel, "Contribution Structures for Requirements Traceability," in *Imperial College of Science, Technology and Medicine*: University of London, 1995.

[9] O. Gotel and A. Finkelstein, "An Analysis of the Requirements Traceability Problem," presented at First International Conference on Requirements Engineering, 1994.

[10] L. Williams, "The XP Programmer: The Few-Minutes Programmer," *IEEE Software*, vol. 20, pp. 16-20, 2003.

[11] R. Jeffries, A. Anderson, and C. Hendrickson, *Extreme Programming Installed*. Upper Saddle River, NJ: Addison-Wesley, 2001.

[12] F. Macias, M. Holcombe, and M. Gheorghe, "A formal experiment comparing extreme programming with traditional software construction," presented at Proceedings of the Fourth Mexican International Conference on Computer Science, 2003.

[13] S. Ambler, "Lessons in Agility from Internet-Based Development," *IEEE Software*, vol. 19, pp. 66 - 73, 2002.

[14] M. Lippert, S. Roock, R. Tunkel, and H. Wolf, "Stabilizing the XP Process Using Specialized Tools," presented at XP 2001, 2001.

[15] P. O'Brian Holt, "HCI tools, methods and information sources," presented at IEE Colloquium on Usability Now, 1991.

[16] J. Nielsen, *Usability engineering*. San Francisco, CA: Morgan Kaufmann Publishers, 1993.

[17] K. Breitman and J. Leite, "Managing User Stories," presented at International Workshop on Time-Constrained Requirements Engineering (TCRE 02), 2002.

[18] J. Nawrocki, M. Jasinski, B. Walter, and A. Wojciechowski, "Extreme Programming Modified: Embrace Requirements Engineering Practices," presented at 10th IEEE Joint International Requirements Engineering Conference, RE'02, Essen, Germany, 2002.

[19] L. Wagner, "Extreme Requirements Engineering," *Cutter IT Journal*, vol. 14, pp. 34-38, 2001.

[20] I. Hooks and K. Farry, *Customer-centered products : creating successful products through smart requirements management*. New York, NY: American Management Association, 2001.

[21] J. Koskela, J. Kääriäinen, and J. Takalo, "Improving Software Configuration Management for Extreme Programming: A Controlled Case Study," presented at European Software Process Improvement, EuroSPI'2003, Graz, Austria, 2003.

[22] F. Maurer and S. Martel, "Process Support for Distributed Extreme Programming Teams," presented at ICSE 2002 Workshop on Global Software Development, 2002.

[23] M. J. Rees, "A feasible user story tool for agile software development?," presented at Ninth Asia-Pacific Software Engineering Conference, 2002.

[24] M. Lippert, S. Roock, and H. Wolf, *Extreme Programming in Action: Practical Experiences from Real World Projects*: John Wiley & Sons Ltd., 2002.

[25] A. Leon, *A Guide to software configuration management*. Boston: Artech House, 2000.

[26] I. Crnkovic, Asklund, U., Dahlqvist, A., *Implementing and Integrating Product Data Management and Software Configuration Management*. London: Artech House, 2003.

[27] A. Sääksvuori and A. Immonen, *Product lifecycle management*. Berlin: Springer-Verlag, 2004.

[28] J. Kääriäinen, J. Koskela, J. Takalo, P. Abrahamsson, and K. Kolehmainen, "Supporting Requirements Engineering in Extreme Programming: Managing User Stories," presented at ICSSEA 2003, Paris, France, 2003.

[29] M. Hult and S.-A. Lennung, "Towards a definition of action research: A note and bibliography," *Journal of Management Studies*, pp. 241-250, May 1980.

[30] G. I. Susman and R. D. Evered, "An Assessment of the Scientific Merits of Action Research," *Administrative Science Quarterly*, vol. 23, pp. 582-603, 1978.

[31] D. Avison, "Action research: a research approach for cooperative work," presented at The 7th International Conference on Computer Supported Cooperative Work in Design, 2002.

[32] D. C. Fowler and P. A. Swatman, "Building information systems development methods: synthesising from a basis in both theory and practice," presented at Australian Software Engineering Conference, 1998.

[33] P. Abrahamsson and J. Koskela, "Extreme programming: A survey of empirical results from a controlled case study," To be presented at ACM-IEEE International Symposium on Empirical Software Engineering (ISESE 2004),, Redondo Beach, CA, USA, 2004.

[34] T. Dingsøyr and G. K. Hanssen, "Extending Agile Methods: Postmortem Reviews as Extended Feedback," presented at 4th International Workshop on Learning Software Organizations, Chicago, Illinois, USA, 2002.

[35] F. Buckley, *Implementing configuration management : hardware, software, and firmware*. Los Alamitos: IEEE Computer Society Press, 1996.

[36] D. Lyon, *Practical CM - Best Configuration Management Practices for the 21st Century*, 2nd ed: RAVEN Publishing Company, 1999.

[37] A. L. Ames, "Users first! An introduction to usability and user-centered design and development for technical information and products," presented at Professional Communication Conference, IPCC 2001, 2001.

Published by

Series title, number and report code of publication

VTT Publications 605
VTT–PUBS–605

Author(s)
Kääriäinen, Jukka

Title

Practical adaptation of configuration management
Three case studies

Abstract

This thesis studies the adaptation of configuration management. Configuration management (CM) is a support process for product development and it operates in the context of the development project. Several factors, such as the size of the project, distribution, development disciplines, etc. affect the project's CM solution. Nowadays, CM practices inside a project have become an industrial de-facto standard, but the complexity emerges from the modern operational environment of product development. Globalisation, outsourcing, product variation and the amount of SW in modern products have characterized the modern product development. This trend has also affected the CM practices, which need to face these new challenges.

The study defines the initial framework of factors that affect the CM solution. These factors represent the project characteristics that the CM adaptation needs to solve when planning the CM solution for a project. The framework of factors has been used to characterise three CM adaptation case studies. The CM practices are considered based on factors in each case and the results are discussed. Furthermore, a cross-case analysis has been carried out to detect and discuss similarities and differences between the cases.

Keywords
software engineering, software configuration management, configuration management, embedded systems, agile methods

ISBN
951–38–6842–7 (soft back ed.)
951–38–6843–5 (URL: http://www.vtt.fi/publications/index.jsp)

Series title and ISSN			Project number
VTT Publications 1235–0621 (soft back ed.) 1455–0849 (URL: http://www.vtt.fi/publications/index.jsp)			E4SU00314
Date June 2006	Language English	Pages 71 p. + app. 48 p.	Price C
Name of project MERLIN (Embedded Systems Engineering in Collaboration)		Commissioned by Tekes, VTT	

Contact
VTT Technical Research Centre of Finland
Kaitoväylä 1, P.O. Box 1100,
FI-90571 OULU, Finland
Phone internat. +358 20 722 111
Fax +358 20 722 2320

Sold by
VTT Technical Research Centre of Finland
P.O.Box 1000,
FI-02044 VTT, Finland
Phone internat. +358 20 722 4404
Fax +358 20 722 4374

VTT PUBLICATIONS

583 Turunen, Erja. Diagnostic tools for HVOF process optimization. 2005. 66 p. + app. 92 p.

584 Measures for improving quality and shape stability of sawn softwood timber during drying and under service conditions. Best Practice Manual to improve straightness of sawn timber. Edited by Veikko Tarvainen. 2005. 149 p.

585 Hyötyläinen, Raimo. Practical interests in theoretical consideration. Constructive methods in the study of the implementation of information systems. 2005. 159 p.

586 Koivisto, Tapio. Developing strategic innovation capability of enterprises. Theoretical and methodological outlines of intervention. 2005. 120 p.

587 Ajanko, Sirke, Moilanen, Antero & Juvonen, Juhani. Kierrätyspolttoaineiden laadunvalvonta. 2005. 59 s.

588 Ebersberger, Bernd. The Impact of Public R&D Funding. 2005. 199 p. + app. 12 p.

589 Kutinlahti, Pirjo. Universities approaching market. Intertwining scientific and entrepreneurial goals. 2005. 187 p. + app. 4 p.

590 Jantunen, Erkki. Indirect multisignal monitoring and diagnosis of drill wear. 2005. 80 p. + app. 110 p.

591 Rauste, Yrjö. Techniques for wide-area mapping of forest biomass using radar data. 2005. 103 p. + app. 77 p.

592 Safety and reliability. Technology theme – Final report. Ed. by Veikko Rouhiainen. 2006. 142 p. + app. 27 p.

593 Oedewald, Pia & Reiman, Teemu. Turvallisuuskriittisten organisaatioiden toiminnan erityispiirteet. 2006. 108 s. + liitt. 10 s.

594 Lyly, Marika. Added ß-glucan as a source of fibre for consumers. 2006. 96 p. + app. 70 p.

595 Hänninen, Saara & Rytkönen, Jorma. Transportation of liquid bulk chemicals by tankers in the Baltic Sea. 2006. 121 p. + app. 30 p.

596 Vähä-Heikkilä, Tauno. MEMS tuning and matching circuits, and millimeter wave on-wafer measurements. 2006. 86 p. + app. 82 p.

597 Lallukka, Sami & Raatikainen, Pertti. Passive Optical Networks. Transport concepts. 2006. 123 p.

598 Lyyränen, Jussi. Particle formation, deposition, and particle induced corrosion in large-scale medium-speed diesel engines. 2006. 72 p. + app. 123 p.

600 Kontkanen, Hanna. Novel steryl esterases as biotechnological tools. 2006. 100 p. + app. 54 p.

601 Askolin, Sanna. Characterization of the *Trichoderma reesei* hydrophobins HFBI and HFBII. 2006. 99 p. + app. 38 p.

602 Rosqvist, Tony, Tuominen, Risto & Sarsama, Janne. Huoltovarmuuden turvaamiseen tähtäävä logistisen järjestelmän riskianalyysimenetelmä. 2006. 68 s. + liitt. 20 s.

605 Kääriäinen, Jukka. Practical adaptation of configuration management. Three case studies. 2006. 71 p. + app. 48 p.

Tätä julkaisua myy	Denna publikation säljs av	This publication is available from
VTT PL 1000 02044 VTT Puh. 020 722 4404 Faksi 020 722 4374	VTT PB 1000 02044 VTT Tel. 020 722 4404 Fax 020 722 4374	VTT P.O. Box 1000 FI-02044 VTT, Finland Phone internat. +358 20 722 4404 Fax +358 20 722 4374

ISBN 951–38–6842–7 (soft back ed.)
ISSN 1235–0621 (soft back ed.)

ISBN 951–38–6843–5 (URL: http://www.vtt.fi/inf/pdf/)
ISSN 1455–0849 (URL: http://www.vtt.fi/inf/pdf/)