VTT PUBLICATIONS 366

Practical development of software configuration management for embedded systems

Jorma Taramaa

VTT Electronics

Academic Dissertation to be presented with the assent of the Faculty of Science, University of Oulu, for the public discussion in the Auditorium L10, Linnanmaa, on November 7th, 1998, at 12 noon.

TECHNICAL RESEARCH CENTRE OF FINLAND
ESPOO 1998

ISBN 951-38-5344-6
ISSN 1235–0621

JULKAISIJA – UTGIVARE – PUBLISHER

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

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

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

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

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

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

Technical editing Leena Ukskoski

Libella Painopalvelu Oy, Espoo 1998

Taramaa, Jorma. Practical development of software configuration management for embedded systems. Espoo 1998. Technical Research Centre of Finland, VTT Publications, 366, 147 p. + 108 p.

Keywords software engineering, software configuration management, embedded software, embedded systems, process improvement, computer programs

Abstract

Software production problems have forced industrial organisations producing embedded systems to assess and change the disciplines used to manage the software process. The development of software configuration management (SCM) is one of the solutions for problems concerning new product features. SCM is also one of the software processes that requires improvements. The importance of SCM is clearly understood by producers of embedded systems, but there are difficulties to find the right procedures to apply and further develop SCM in practice.

This thesis introduces a descriptive framework for SCM as a part of development of more comprehensive software engineering practices of embedded systems. The SCM framework covers version control, release-oriented and change-oriented SCM. In addition to these SCM-specific procedures, there are CM solutions for other product technologies and for global product management including distribution.

The SCM framework has been evaluated and applied in co-operation with several industrial embedded systems manufacturers in the field of mechatronics, space instruments and other electronics applications. The framework can be regarded as a starting point for the further development of a SCM maturity model.

The tentative SCM maturity improvement levels are associated with the SCM elements and related logistics processes, such as order/delivery and customer data management. The maturity levels of the improvement range from low-level version control to global product management including parallel solutions of SCM and related elements.

The improvement of the SCM process calls for a maturity assessment and improvement procedure. This research describes an inductive procedure, PR^{2}IMER, used in the first experiments as a part of the incremental approach.

Preface

This research was carried out at VTT Electronics. The foundation for the work was laid during my stay at the University of Maryland at College Park in 1989 - 1990. The results of this visit produced the idea of constructing a reverse engineering tool for PL/M code. The tool was implemented in the ARKEO project in 1991 - 1993. This work also provided a more profound viewpoint on software maintenance and its relation to software configuration management.

The Finnish project LOKKI, carried out in 1993 - 1994, addressed configuration management in mechatronics applications, where the role of software is not as central as in some other applications. A prototype of a software configuration management system was built that indicated a need to proceed gradually from version control to configuration management and maintenance.

The main context of this research was the ESPRIT project AMES, carried out in 1993 - 1996, which produced a set of tools, such as disabbreviation and domain analysis tools, as parts of an application understanding toolset. The concept 'application management' that was used in modelling the application management process led to a comprehensive view of the needs of this work.

The Finnish project LEIVO, which took place in 1994 - 1996, resulted in co-operation with industry for proceeding incrementally towards developing configuration management practices. Experiences with this project strengthened the need for the use of various configuration management steps when developing software configuration management practices in each company.

In the final phase of the research, in 1995 - 1997, the European Space Agency's (ESA/ESTEC) PMod project provided additional support for the evaluation of software configuration management solutions in space applications.

I would like to thank my colleagues at VTT Electronics: Dr. Antti Auer, Mr. Jukka Korhonen, Dr. Kari Laitinen, Mr. Raino Lintulampi, Ms. Minna Mäkäräinen, Dr. Markku Oivo, Ms. Heli Puustinen, Mr. Hannu Rytilä, and Mr. Matias Vierimaa. I also want to be grateful to all my co-operation partners of other organisations, Dr. James Purtilo of the University of Maryland, Mr. Esa Tuovinen of Honeywell Automation, Mr. Jussi Eronen of Datex-

Instrumentarium, Mr. Ari Tuunanen and Mr. Arto Sirén of Tamrock Drills, Mr. Tommi Ketola, Mr. Kari Kumpulainen and Ms. Kati Suominen of Space Systems Finland. In addition, I am also grateful to all the people in the AMES project, which provided an excellent forum to exchange thoughts of the topics of my dissertation.

Prof. Veikko Seppänen read several drafts of the dissertation and provided several valuable comments. In addition, the co-operation with Prof. Seppänen has been excellent for all these years. Mr. Malcom Hicks proof-read the dissertation. Mr. Douglas Foxvog has checked my language in most of my scientific papers. My sincerest thanks for them all.

I also wish to thank Prof. Reidar Conradi and Dr. Pekka Isomursu, the reviewers of this dissertation, for their valuable and constructive comments, which have led to a better construction of my dissertation. The work has been supervised by Prof. Samuli Saukkonen. I wish to thank him for guidance and comments.

The research projects behind this work have been financed by TEKES, VTT Electronics, and several companies. I am grateful for their support.

Finally, I wish to express my deepest gratitude to my family, Raija, Sakari and Mirjami, for their support and patience to the family member who is always sitting alone in the workroom.

Oulu, October 1998

Jorma Taramaa

List of original papers

This thesis includes seven original papers published or accepted for publication in scientific journals or the proceedings of international conferences. They are included here with the permission of their original publishers.

I Taramaa, J. and Oivo, M. Evaluation of Software Maintenance of Embedded Computer Systems. In: D.W. Russell (ed.), International Symposium on Engineered Software Systems, Malvern, Pennsylvania, May 1993. Published by World Scientific Publishing Co., Singapore. Pp. 193 - 203.

II Taramaa, J. and Rytilä, H. A Solution of Reverse Engineering Process Using Different Knowledge Categories for Preventive Maintenance. Proceedings of the Seventh European Software Maintenance Workshop, Durham, UK, September 1993. 10 p.

III Taramaa, J., Lintulampi, R. and Seppänen, V. Automated Assembly of Machine Control Software. Mechatronics 1994. Vol. 4, No. 7, pp. 753 - 769.

IV Taramaa, J., Mäkäräinen, M. and Ketola, T. Improving Application Management Process through Qualitative Framework. In: G.-L. Caldiera and K. Bennett (eds.), Proceedings of the International Conference on Software Maintenance (ICSM'95), Opio (Nice), France, October 1995. Published by IEEE Computer Society Press, Los Alamitos, California. Pp. 327 - 336.

V Taramaa, J., Seppänen, V. and Mäkäräinen, M. From Software Configuration to Application Management - Improving the Maturity of the Maintenance of Embedded Software. Journal of Software Maintenance: Research and Practice 1996. Vol. 8, No. 1, pp. 49 - 75.

VI Auer, A. and Taramaa, J. Experience Report on the Maturity of Configuration Management for Embedded Software. In: I. Sommerville (ed.), Proceedings of the 6th International Workshop on Software

Configuration Management (SCM6), Berlin, Germany, March 1996. Published by Lecture Notes in Computer Science 1167, Springer Verlag, Heidelberg, Germany, 1996. Pp. 187 - 197.

VII Vierimaa, M., Taramaa, J., Puustinen, H., Suominen, K. and Ketola, T. Framework for Tool Evaluation for a Maintenance Environment. Journal of Software Maintenance: Research and Practice 1998. Vol. 10, No. 3, pp. 203 - 224.

The author of this thesis is the principal author of the other papers than Paper VI and VII. In both of these papers the author's effort has been, however, essential. Paper I indicates the role of software maintenance. Paper II provides a solution for reverse engineering as part of software maintenance. Paper III introduces an approach to build a software configuration management system. Paper V extends the point of view from configuration to application management, taking into account software maintenance aspects. Paper IV further relates application management to process modelling. Paper VI provides experimental results of industrial application the framework presented in Paper III and extended in Paper V. Paper VII shows the results of the quantitative validation of the tools proposed for the application management.

Contents

Appendices of this publication are not included in the PDF version. Please order the printed version to get the complete publication (http://www.inf.vtt.fi/pdf/publications/1998)

List of acronyms

AMES	Application Management Environments and Support, a project of the ESPRIT programme #8156
ARKEO	Reverse engineering technology for embedded software, a Finnish information technology project funded by TEKES and Finnish industry and VTT Electronics
AU	Application Understanding toolset, a part of the AMES tools, developed by the University of Durham and VTT Electronics
ASIC	Application Specific Integrated Circuit, an implementation technology of complex logic functions in hardware
BIOS	Basic Input/Output System, a set of instructions stored on a ROM chip inside IBM PCs and PC-compatibles, which handles all input-output functions.
BOM	Bill-of-Material
BOOTSTRAP	European software process assessment and improvement methodology
CAD	Computer-Aided Design
CAE	Computer-Aided Engineering
CAM	Computer-Aided Manufacturing
CASE	Computer-Aided Software Engineering
CI	Configuration Item
ClearCase	SCM tool, developed by Rational Software Corporation Inc.
CLU	Object-oriented programming language developed at MIT
CM	Configuration Management

CMM	Capability Maturity Model for software processes, an American software process assessment and improvement programme and framework
Continuus	SCM tool, developed by Continuus Software Corporation
CP	Configuration Programming
DARPA	Defense Advanced Research Projects Agency
DoD	Department of Defence
DSP	Digital Signal Processing
EPROM	Erasable Programmable ROM
EPSOM	European Platform for SOftware Maintenance, a subproject of ESF
ESA	European Space Agency
ESF	Eureka Software Factory
ESPRIT	European Strategic Programme for Research and development in Information Technology
ESSDE	European Space Software Development Environment
ESSI	European Software System Initiative
ESTEC	European Space research and TEChnology Centre at Noordwijk, the Netherlands
EU	European Union
Eureka	Pan-European, market-oriented research and development programme
EXPRESS	Language for data description related to STEP standard
FPGA	Field Programmable Gate Arrays (FPGAs), an implementation technology of complex logic functions in hardware
GQM	Goal-Question-Metric
HOOD	Hierarchical Object Oriented Design
HTML	HyperText mark-up language

IAS	Impact Analysis toolset, a part of the AMES project tools, developed by Matra Marconi Space, France
IDEF0	Technique for specifying functional relationships, a commercialised solution of SADT
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Standardization Organization
LEIVO	Practical improvement of software configuration and version control, a Finnish information technology project funded by TEKES and Finnish industry
LOKKI	Assembly of machine control software, a Finnish information technology project funded by TEKES, Finnish industry and VTT
MIL	Module Interconnection Language
MRP	Material Requirement Planning
MS-ACCESS	Database, developed by Microsoft Inc.
OVUM	Independent research and consulting company
PDM	Product Data Management
PL/M	Programming Language for Intel 80x86 processors
PMod	Study on Process Models for Space Applications, a project of ESA's Technology Research and Development Programme
Polymake	Configuration Builder, produced by Intersolv Inc.
PSS-05	ESA's software engineering standard
PR^{2}IMER	Practical Process Improvement for Embedded Real-Time Software, developed by VTT Electronics
ProcessWEAVER	Process enactment tool, developed by Cap Gemini Innovation
PROM	Programmable ROM

PROMESSE	PROcess Modelling in ESSDE, a project of ESA's Technology Research and Development Programme
PROTEUS	Support for System Evolution, a project of the ESPRIT programme #6086
PVCS	Professional Version Control System, developed by Intesolv Inc.
RCS	Revision Control System, a version control system developed by prof. Walter Tichy in Carnegie Mellon, Pittsburgh, Pennsylvania
REDO	Maintenance, Validation and Documentation of Software Systems, a project of the ESPRIT programme #2487
ReverseNICE	Reverse-engineering tool, a part of the AMES project tools, developed by Intecs Sistemi Spa
Robochart	Flow-diagram editor, produced by Digital Insight Inc.
ROM	Read Only Memory
RT-SA/SD	Real Time Extensions for Structured Analysis and Design is a software engineering methodology
SADT	Structured Analysis and Design Technique
SCCS	Source Code Control System for UNIX systems
SCM	Software Configuration Management
SCM-PR^{2}IMER	SCM-focused process improvement approach of PR^{2}IMER
SME	Small or Medium Enterprise
SPI	Software Process Improvement
SPICE	Software Process Improvement and Capability dEtermination
STEP	ISO's STandard for the Exchange of Product data
Tcl/Tk	Scripting language for applications developed to run on Sun Microsystems environments

TDIF	Transfer Data Interface, a traceability data input format, defined by the AMES project
TEKES	Technical Development Centre of Finland
TPM	Total Product Management in Technopolis, a project of the EU/ESSI programme
TRILLIUM	Software process assessment and improvement methodology developed by Northern Telecom and Bell Canada
TQM	Total Quality Management
UNIX	Operating System, trademark of AT&T
Webmaker	FrameMaker cross-reference converter, developed by Harlequin Group Limited
VHDL	Hardware description language, originally developed by DARPA.
VISCOUNT	Virtual Software Corporation Testbed, a project of the ESPRIT programme #25754
VTT	Technical Research Centre of Finland
XCOPY	PC/DOS copying command

1. Introduction

The software of embedded computer systems, embedded software, consists of built-in computer programs for controlling high value-added products such as switching and production control systems, space instruments, wireless communication devices, home electronics goods, and mechatronic machines. Definitions and taxonomies of embedded software are given by Karjalainen (1987), Seppänen (1990) and George and Kryal (1996), for example.

The complexity of embedded software has vastly increased during the last few years, as also has its size, reaching hundreds of thousands of lines of code in some applications (Davis 1990; Hurst and Dennis 1996; Seppänen et al. 1996). The evolution of embedded software in consecutive generations of mobile phones in terms of typical size is shown as an example in Table 1 (Karjalainen et al. 1996). The analogue-based solutions were implemented in 1980's and digital ones in 1990's. This development is also indicated by analysis of telecommunication products where as much as 75 per cent of the product's development costs was already directed at software in the beginning of 1990's (Weider et al. 1990).

Table 1: Increase of software size in mobile phones.

System type	Generation	Example of system	Software size
Analogue	1st	Nordic mobile phone system (NMT)	some Kbytes
Analogue	2nd	NMT	tens of Kbytes
Digital	1st	Global system for mobile telecommunications (GSM)	hundreds of Kbytes
Digital	2nd	GSM	about 1 million bytes

An example of the direction of a specific application software is indicated by the European Space Agency's (ESA) on-board software development. The first spacecraft in 1970's included very simple and small control software, whereas the current modern spacecrafts rely heavily on software, i.e. measurement instruments, on-board control and telecommunication (Mortensen 1995).

Products based on embedded software are typically used for a long time and require several modernisation cycles. A state of the art study concerning the maintenance of embedded software indicated that the current software generation will be used in many cases almost throughout the 1990's (Paper I). The average age of embedded software in active use was about seven years, and there were some items that had been used more than for twenty years.

In addition to the increase in software size and complexity, many companies have faced a new situation in which software has come to occupy a central role among their products.

The products based on embedded computer systems can be characterised by the following changes in their features:

- The volume of products supplied has increased.
- Software belonging to the same product family may be used in a number of products.
- There may be several versions of the same product, representing the outcomes of different tool versions and possessing slightly different features.
- The products are integrated as parts of various other systems.
- The volume of changes in the products has increased.

1.1 Motivation and scope of research

Software production problems have forced industrial organisations to assess and change the disciplines used to manage the software process (Sommerville 1995; Pressman 1996). Good examples of this development are provided by well-known quality certification standard based on ISO 9000-3 (ISO 1990-3 1991), total quality management (TQM) framework (Deming 1982) as well as software process assessment and improvement programmes such as CMM (Humphrey 1988; Paulk et al. 1995), TRILLIUM (Bell 1994), BOOTSTRAP (Kuvaja et al. 1994) and SPICE (ISO/IEC 15504 1995). Software process assessment and improvement programmes also include a classification of software processes, although each classification is slightly different. In all of them software configuration management (SCM) is an independent process closely related to many other processes.

The purpose of SCM is to manage the evolution of large and complex software systems by establishing and maintaining the integrity of software throughout its life cycle. This means identifying the configuration of the software, systematically controlling changes of the configuration, and maintaining traceability of the configuration throughout its life cycle (Dart 1991; Tichy 1988).

It has been understood that SCM requires improvements as one of the software processes (Berlack 1992; Buckley 1996; Forte 1993; Tichy 1988; Whitgift 1991; Dart 1992a). Its importance is clearly understood in industry, but there are difficulties in finding the procedures for adopting and improving it (Brown et al. 1995; Seppänen et al. 1994; Seppänen et al. 1996).

The change of the position of SCM is indicated by the sales of commercial SCM tools and environments. According to OVUM (Rigg et al. 1995), the annual rate of growth in SCM revenues is 50% and the market will rise to almost $1 billion by 1997. SCM has long been accepted as one of the activities contributing to software engineering, but it has proved to be difficult to apply effectively until recently (Rigg et al. 1995). The field of SCM typically discusses general technical solutions (Forte 1993; Rigg et al. 1995).

Although commercial SCM solutions exist, there are specific dimensions, which must be studied, and for which solutions need to be developed, e.g. new life cycle models (Davis and Bershoff 1991) and architectural questions (MacKay 1995; van der Hoek et al. 1996).

Incremental SCM improvement, where companies improve their SCM practices in a stepwise manner, can be also regarded as a specific category. Ambriola et al. (1990), for example, discuss SCM generations that have been produced by different software development environments.

Although automatisation opportunities of SCM are understood fairly well today, there is a need to evaluate the adoption of automated configuration tools (Dart 1995; Wein et al. 1995). Dart (1995; 1996) presents a framework that includes a number of issues which have to be taken into account, such as technical, political, risk-related and organisational issues. Our approach concentrates on technical issues, since these form a distinct improvement target. Technical issues have to be

understood before it is possible to take other issues into account. The software production organisation must also assess political, risk-related and personnel issues, although this research does not deal with them.

We have developed an approach to SCM development, which has been evaluated and applied in co-operation with several industrial embedded systems manufacturers. The importance of SCM development arises out of the characteristics of the companies (Papers I and VI). The development groups are typically small in size (5 - 20 developers) which does not make it possible to invest large amounts of effort in software process development. In addition, existing software engineering environment solutions, including SCM, force companies to make incremental steps in parallel with their normal software development.

1.2 Research problem

SCM and software maintenance have been extensively discussed in the literature (e.g. Arthur 1988; Bennett et al. 1991; Berlack 1992; Bershoff et al. 1979; Dart 1990 and 1991; Feiler 1991; Forte 1993; Lientz and Swanson 1980; Pressman 1996; Sommerville 1995; Tichy 1988; Whitgift 1991). Some authors in particular have taken embedded systems and their software as a specific SCM category, notably Adamson (1990), Buckley (1996), Foster et al. (1989) and Lyon (1997).

The hardware of embedded computer systems makes the changes slightly different from those applying to software based on standard workstation and PC hardware. Software production and product manufacturing are closely related, because the embedded software is one part of the physical product to be manufactured and supplied. This means that SCM of embedded software has to be seen within a more comprehensive framework, which includes interfaces with hardware and other technologies as well as software production as a part of product development. The changes made after delivery are typically corrections, giving rise to economic losses in addition to other potential problems such as safety and security concerns (Leveson 1986; Liddiard 1994).

Changes made to embedded software can be regarded more as questions of evolution than of maintenance. In addition to normal corrective actions, new software is often developed by modifying existing configurations. The evolution is then both adaptive

and perfective. Adaptive maintenance relates software to environmental changes, whereas perfective maintenance takes functional changes into account (Swanson 1976).

From the previous discussion we derive the research problem as follows:

How can an industrial organisation cope with software configuration management of embedded software taking into account the requirements set by product evolution and maintenance?

Based on this statement of the topic, the following more specific questions can be put forward:

Q1. What is the role of software evolution and maintenance in products which include embedded software?

Q2 What activities are needed for managing configurations of embedded software?

Q3 How is it possible to analyse SCM development of embedded systems?

Q4 How is it possible to proceed in SCM improvement?

1.3 Research methods

The approach adopted here follows a constructive research model that includes both conceptual and technical realisations. Since this research concentrates on constructing the framework and on the further improvement of SCM, this approach can be regarded as constructive. In addition, the work includes separate technical solutions, which extend the research approach in a technical direction.

Such a method represents one of the three approaches recognised in the context of information systems development by Iivari (1991)[1] .

The approach of this research can be also examined in a more comprehensive framework. In terms of the classification of research methods presented by Pertti and Annikki Järvinen (1995)[2] it can be characterised as a combination of the theory building method and the construction of a new reality. The former includes analysing SCM practices and the latter the technical solutions used for the various SCM environments.

Our assumption is that there is a need to develop software engineering processes, in which SCM constitutes one of the targets to be developed. In addition, software maintenance and evolution include special features upon which embedded software has its own effects.

As depicted in Figure 1, the research method can be divided into the following three phases:

1. **Problem analysis:** We analyse state-of-the-art practices in SCM and software maintenance, taking the special features of embedded software into account. Software maintenance is chosen as the main point of view in process modelling. Product configuration practices and programming languages add elements of their own to the research topic.

---

[1] According to Iivari's categorisation, there are three major research methods of information systems: *constructive* (including conceptual and technical development), *nomothetic* (including formal-mathematic analysis, experiments, and field studies), and *idiographic* (including cases and action research) ones.

[2] Pertti and Annikki Järvinen describe research work in five major categories. They first separate *mathematical research* from the approaches intended to study the reality. The latter is then further divided to *conceptual-theoretical research* and empirical research. The empirical approaches can concern *construction of a new reality* or a past and present one. The latter can still be considered with two different approaches: with *theory testing methods* or *with theory building methods*.

2. **Construction:** An approach to the development of a descriptive framework for SCM is suggested. In addition, this thesis introduces the tentative improvement levels to SCM as a starting point for the further development of a maturity model. Both are justified by the results of the problem analysis.

3. **Demonstration**: The solution is produced and validated by case studies in three projects. They include various kinds of embedded systems applications, such as space instrument, mechatronic system, and other electronic products. The SCM framework validation is made by these three applications. In addition, the tentative SCM maturity improvement levels associated with SCM elements are validated by two of the SCM framework validation projects that include space instrument and electronic products.

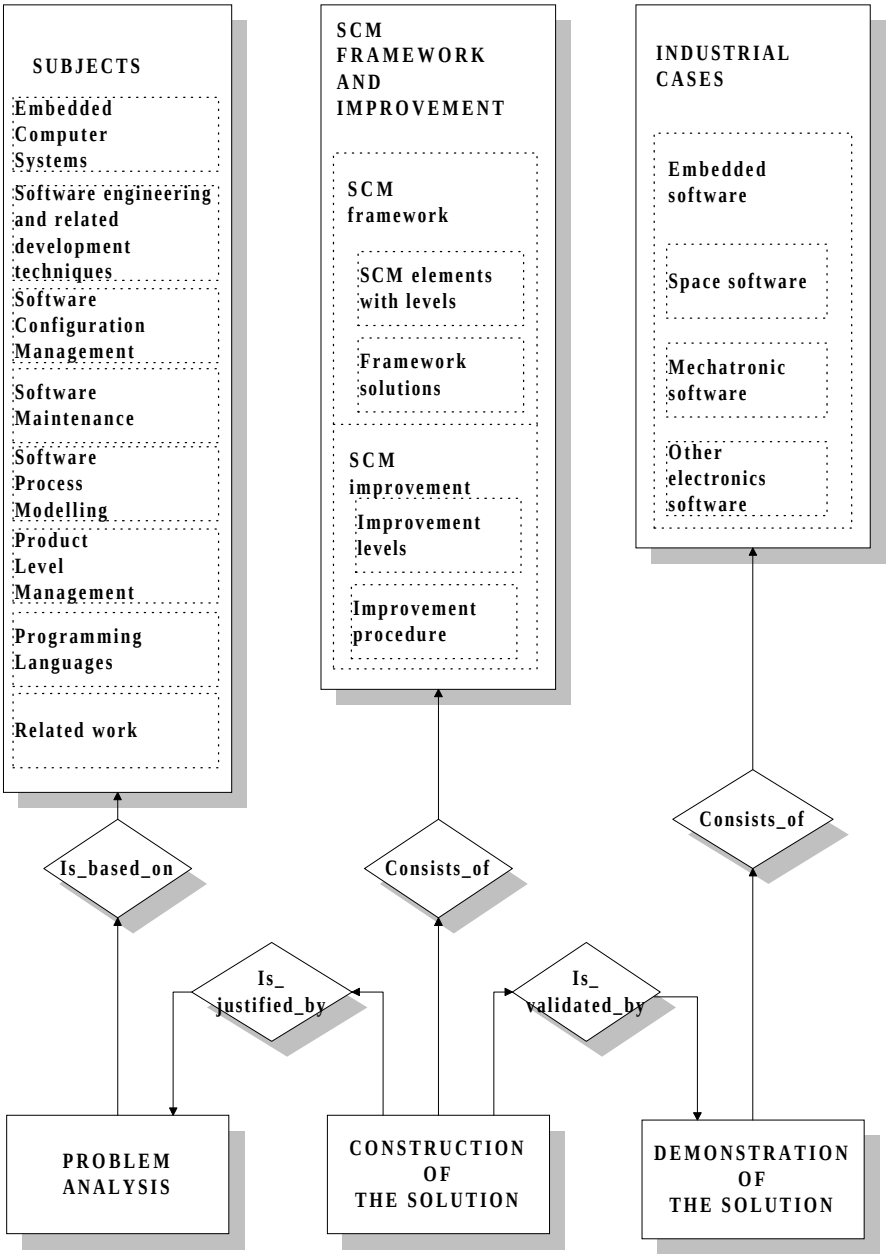

Figure 1. The research method.

Since this thesis is based on previous experiences of SCM and its application in industry and on several projects, their relations to this thesis are described by Table 2.

Table 2: Projects and their results related to this thesis.

Time	Description of research activity	Results	Position in this thesis	References
1985 - 90	Industrial projects at VTT Electronics	Experiences about SCM and its position in the development of embedded software	Industrial experiences, references in chapters 4 and 5	
1990	Module interconnection technology and its relation to embedded software	Understanding of the module interconnection technology	References of section 2.2	Taramaa and Purtilo 1990
1992 - 93	The ARKEO project: Reverse engineering technology as a part of software maintenance of embedded systems	Position of software engineering processes in the companies producing embedded software Reverse engineering solutions for embedded software	Original paper I of this thesis Original paper II of this thesis References in chapter 2	Paper I Paper II; Rytilä 1994
1993 - 94	The LOKKI project: SCM for mechatronics applications	A SCM environment definition and construction Preliminary experiences of the relation between SCM and PDM	Original papers III and V of this thesis Requirements of section 3.1 Validation results of chapter 6	Papers III and V

Time	Description of research activity	Results	Position in this thesis	References
1994 - 96	The AMES project: Application management development – space software as a case	Methodology, environment and tool development for application management The first experiences of SCM framework and incremental improvement procedure	Original papers IV, V and VII of this thesis References in section 2.4 Requirements of section 3.2 Validation results of chapter 6	Papers IV, V and VII; Mäkäräinen and Taramaa 1995; Taramaa and Seppänen 1995; Taramaa and Ketola 1995
1994 - 96	The LEIVO project: Incremental development of software configuration practices	Experiences of SCM and tentative maturity improvement levels	Original paper VI of this thesis Requirements of section 3.3 Validation results of chapter 6	Paper VI
1995 - 97	The PMod project: Evaluation process descriptions for the ESA's software engineering solutions including SCM	Analysis and description of SCM as a part of the more comprehensive software process description	Section 2.6.2	Taramaa 1997; Stragapede et al. 1997

1.4 Focus of the thesis

The approach taken in this thesis concentrates on SCM in embedded systems, relating the specific requirements and features of embedded software evolution and maintenance to SCM. The main focus is on presenting a framework for use when developing SCM for embedded computer systems. The framework takes into account the special features imposed on SCM by these systems.

The presentation of the framework becomes difficult without any definition of the SCM and related concepts. Therefore this thesis describes the basic SCM concepts, maintenance process and its relations to SCM as well as more comprehensive notion of application management which is related to software maintenance and SCM. In addition, the relation to product data management is described for indicating the need to relate SCM and PDM to each other. The construction of SCM and application management environments has provided experimental results, which have been applied when specifying the SCM framework and improvement. The results of the construction of these environments are depicted in Papers III and V.

The main part of validation concentrates on the SCM framework. It is evaluated and applied in several industrial embedded systems applications in the field of mechatronics, space and other electronics applications. The first results regarding this framework are presented in Paper V, which deals with mechatronics and space applications. The framework is applied more comprehensively in Paper VI where the companies ranging from Small and Medium size Enterprises (SME) to big companies with their dedicated quality organisations provided electronics applications at different maturity levels of improvement.

The tentative SCM maturity improvement levels associated with SCM elements, such as version control, software manufacturing and change control, and related logistic processes. The validation was based on two projects. The first project was focused on use of qualitative analysis of current practices, GQM (Goal-Question-Metric) for goals definition, quantitative measurements derived from GQM plan. The validation in the other project was based on qualitative evaluation. The maturity levels of improvement range from low-level version

control to global product management including parallel solutions of SCM elements.

1.5 Outline of the thesis

In the second chapter of this thesis we discuss SCM and maintenance-related activities. Maintenance aspects are emphasised, because they set particular requirements for SCM. The related work is also dealt with by this chapter. SCM in different software process assessment paradigms is analysed, such as CMM, TRILLIUM, BOOTSTRAP and SPICE, as well European Space Agency's (ESA) software maintenance and SCM practices. In addition, some other SCM specific approaches will be discussed.

The requirements set by the industrial partners are the focus of the third chapter indicating the challenges set by these applications, i.e. mechatronics, space and electronics applications.

The focus of the forth chapter is to provide a descriptive framework for SCM as a part of development of more comprehensive software engineering practices. The SCM framework covers version control, release-oriented and change-oriented SCM. In addition to these SCM-specific procedures, there are CM solutions for other product technologies and for global product management including distribution.

The fifth chapter introduces the tentative SCM maturity improvement levels associated with SCM elements, such as version control, software manufacturing and change control, and related logistic processes. The improvement of the SCM process needs a maturity assessment and improvement procedure. The PR^{2}IMER (*Pr*actical *Pr*ocess *Im*provement for *E*mbedded *R*eal-Time Software) framework is used as an improvement procedure of SCM. The SCM framework acts as a descriptive model for the current state analysis supporting the understanding and analysis of SCM practices. The GQM based approach is related as a part of PR^{2}IMER for defining metrics.

The sixth chapter illustrates the first experiences of both SCM framework validation and SCM improvement. The quantitative results have been presented from the change control-oriented improvement part, where the tools developed for application management were validated using GQM. The qualitative analysis was carried out in version control, software manufacturing and order/delivery data management.

A more profound description is given by original papers. From them there is an introduction to the original papers included in chapter seven of this thesis. As an introduction to the papers, we consider the role and state-of-the-art of software maintenance of embedded systems (Paper I). Paper II shows the first results developed for software maintenance, in particular, problem understanding. After the definition and experiments of the SCM environment (Papers III, V and VI), we extend our perspective to the software evolution and maintenance process (Paper IV). Industrial cases are described in Papers III - VI. Paper IV deals with a space instrument, Paper III with a mechatronic system, and Paper VI with experiences gained in several other electronics applications and the first experiences of the tentative maturity improvement procedure. Paper V deals with a combination of space and mechatronics applications. Paper VII shows the results of the quantitative validation done to the tools of the application management environment as a part of the improvement procedure.

Table 3 shows in which way and in which chapter each research question is described. The papers are also related to research questions.

Table 3: Questions and research method and results related to them.

Research question	**Description of research activity**	**Description of research method and results**	**Chapters and papers**
Q1	Relate the role of software evolution and maintenance with SCM	Questionnaire and its analysis **Results:** The role of software maintenance and evolution	Chapter 2 and the results in Papers I and II
Q2	Investigate activities that can be separated in SCM	Analysis of references **Results:** SCM concepts to be used in this research	Chapter 2 and the first results in Paper III
Q3	Develop SCM framework for developing software evolution and maintenance practices and provide evidence of its usefulness	Construction of a new reality of SCM development **Results:** SCM framework including levels of SCM elements	Chapters 4 and 6 as well as solutions shown in Paper V
Q4	Investigate the SCM improvement and provide the first experiments of its usefulness	Application of a process improvement to SCM **Results:** SCM improvement with improvement procedure, program and solutions	Chapters 5 and 6 as well as experiences shown in Papers IV, VI and VII

2. Problem analysis

This chapter provides a background of the technologies to be related to CM, such as embedded systems, software engineering of embedded systems, SCM itself and related concepts. In addition, the comparison of the related work is provided at the end of this chapter.

Although software development for embedded systems involves a lot of specific features, the basic embedded software development tasks are the same as in software engineering in general. Embedded systems are seen as the domain to be analysed in the relation to SCM. Therefore the introduction of SCM and change related activities are divided into general and embedded software-specific parts.

Process modelling related to software maintenance and SCM is one of the topics of this chapter. The process aspect can be regarded as an element unifying software maintenance and SCM, i.e. software maintenance can be supported by process definitions and descriptions, taking into account different maintenance types that require different software processes, such as SCM.

The concepts that we are dealing with in the following sections are broadly set out in Figure 2, which describes the relationships between embedded software, life cycle concepts and process modelling.

2.1 Domain knowledge related to SCM

In this section embedded systems establish domain which is analysed from different viewpoints. Embedded systems bring the specific features to SCM which can be analysed in the target to be developed, i.e. embedded computer system, and in the engineering technology to be applied, i.e. software engineering.

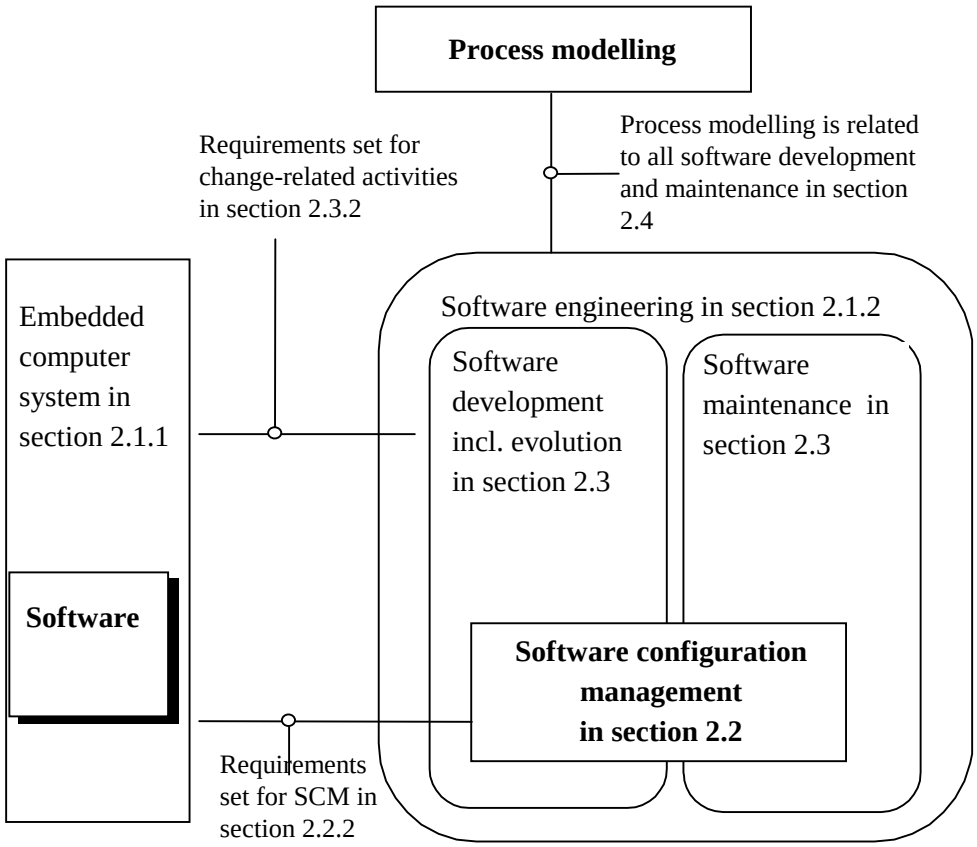

Figure 2. The description of the concepts defined.

2.1.1 Embedded computer systems

Embedded computer systems (referred to as embedded systems) are products, which are directly incorporated into electromechanical devices other than general-purpose computer hardware, devices known as target systems. Their primary purpose is to control the target system, very often within strict timing constraints (Stankovic 1996).

In Figure 3 an embedded system, such as electronic product, is described as a component of an environment which includes users, the target system to be controlled and other systems with which there is need for communication.

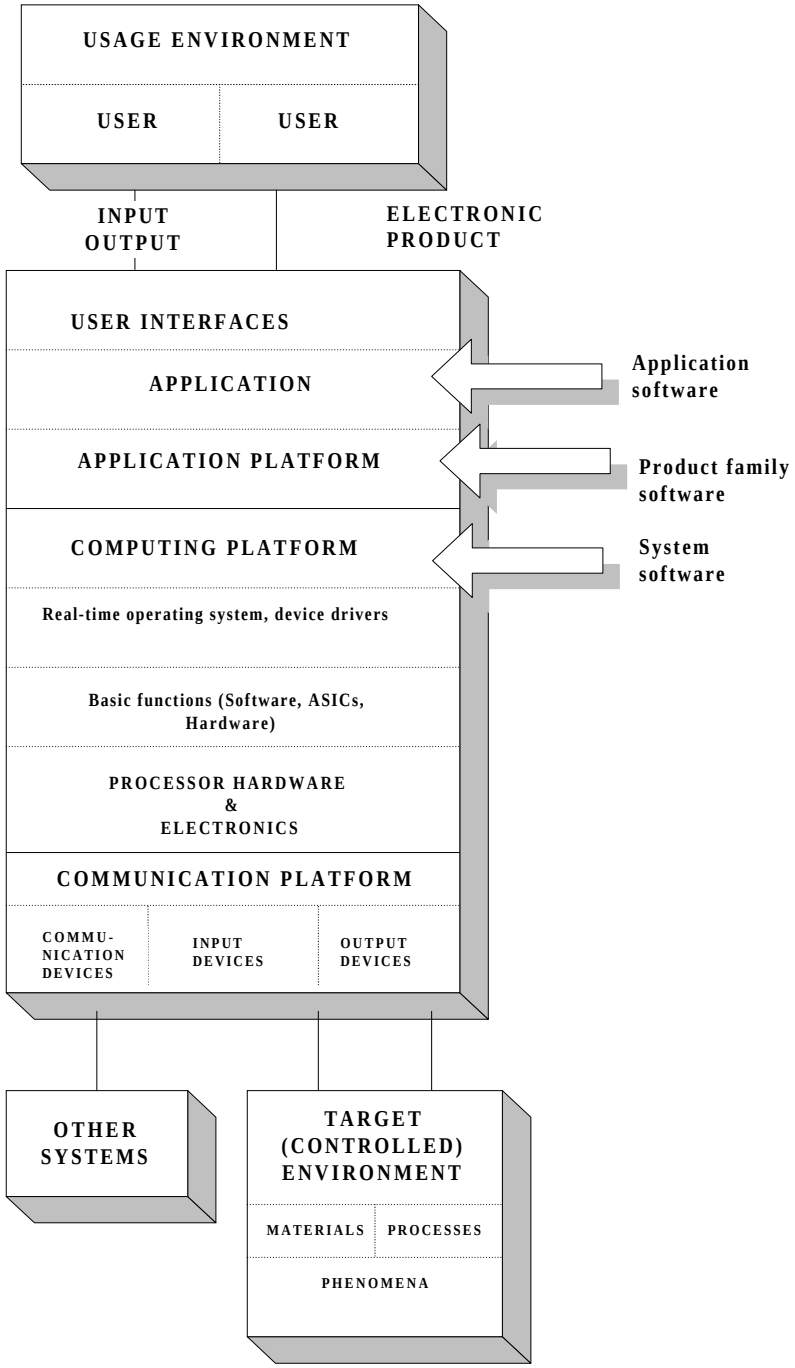

Figure 3. Embedded computer systems and their software categories.

In addition to the traditional control activities, embedded systems are also more dedicated to data communication and processing in various telecommunication applications (Hurst and Dennis 1996; De Micheli 1996).

The importance of embedded systems is indicated by the volume of the world's total processor technology used for embedded applications. In 1995 more than 1.6 billion central processing chips were supplied for embedded computing applications as compared with just 20 million for PC use. In addition, annual growth in the future is estimated to be 20% per year (George and Kryal 1996).

The embedded system is further divided into main components:

- *application platform* including user interface and application software,
- *computing platform* including system software and hardware, such as ASICs and FPGAs, and processor solutions with other electronics, and
- *communication platform for external communication.*

Application functions of application platform are executed on a computing platform by a wide variety of processing devices, such as processor hardware, or automata based on FPGAs or ASICs. Processor hardware is ranging from microcontrollers to 32-bit processors and special-processors, such as digital signal processors (DSP). Software-based solutions are executed on processor hardware, which uses other electronics, which includes e.g. data memories, and buses for supporting information processing executed by computing platform.

Embedded software falls into at least three major classes (Figure 4):

- *system software*, including the operating system, communication, device control and platform-specific functions (e.g. initialisation tests, memory loader, debugging aids etc. by which it is possible to deal with hardware problems),
- *product family software* not unique to the application (to be used in products of several kinds), and
- *application software* (specific to one application).

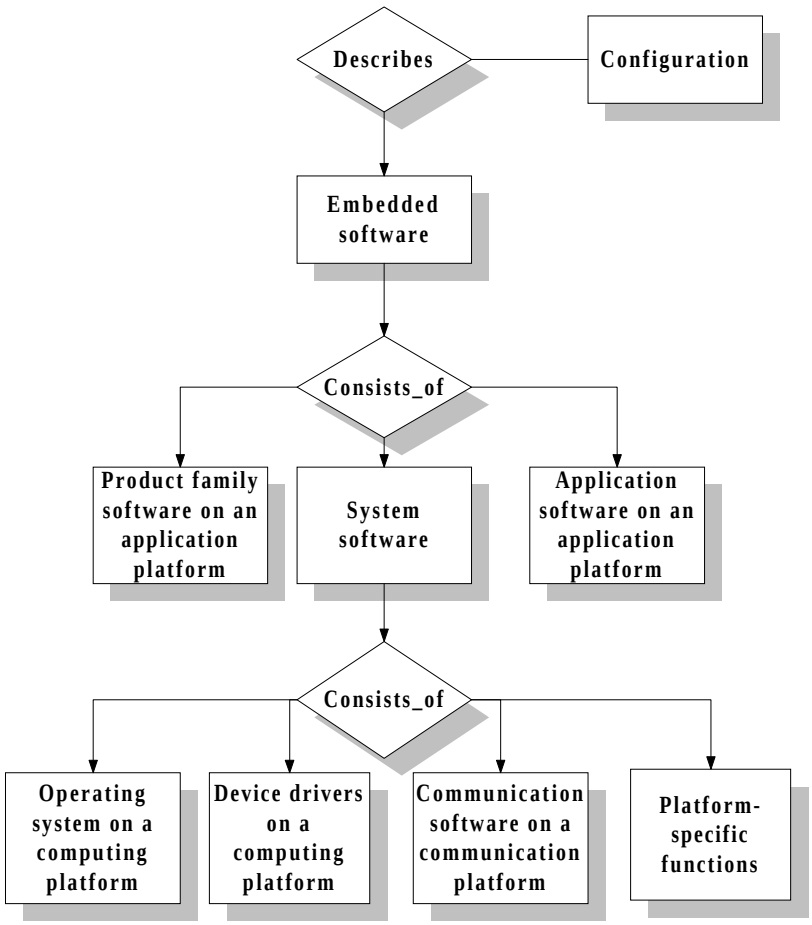

Figure 4. Typical configuration of embedded software.

This taxonomy provides one way of distinguishing maintenance concerns, as follows (Figure 3):

System software may be so specific that a change in the operating system or hardware might require extensive modifications or a total rewrite of other code.

1. A new version of non-unique product family software may be incompatible with the rest of the software.
2. Application software may become incompatible with the special system software and hardware when new computing or communication platforms are taken in use.

2.1.2 Software engineering and related development techniques of embedded systems

Software engineering is defined as a systematic approach to the analysis, design, implementation and maintenance of software. Software development is typically described by different paradigms, which show the steps or stages in the development of software (Boehm 1976; Agresti 1986; Boehm 1988; Sommerville 1995; Pressman 1996).

Because of the essential role of software in embedded systems, the importance of software engineering has been observed as an essential factor of the embedded system to be developed and maintained. Embedded systems and their several product technologies bring the own specific features to these paradigms. Hardware/software co-design and concurrent engineering have been seen as the most essential special features of embedded systems development. Hardware/software co-design indicates the close relation in these technologies and concurrent engineering tells the parallelism in the development of different product technologies (Vidovic and Ready 1993; Savola et al. 1995; De Micheli 1996; Heikkinen 1997).

Therefore, software engineering of embedded systems includes specific features, such as (Adamson 1990; Seppänen 1990):

- Software is one of the implementation technologies. Software-based solution might be alternative to the hardware solution.
- Processor hardware resources can be limited because of big production series and the weight and power requirements. This then impacts software implementation opportunities.
- Tight real-time, safety and security requirements demand specific computer-aided software engineering (CASE) tools.

2.2 Configuration management in software engineering

2.2.1 General principles of SCM

Configuration management (CM) is a discipline for organising and controlling evolving systems. CM has its roots in the aerospace industry of the 1950s, where it was used for guaranteeing the reproducibility of spacecraft (Tichy 1994). Nowadays product-level CM is an area of active research, as indicated by Estublier et al. (1998), Tiihonen et al. (1995), and Westfechtel and Conradi (1998).

SCM can be regarded as a special case of general CM, being distinguished from it in two ways (Tichy 1988): software is usually easier and faster to change than hardware, and SCM is potentially more automatable because of computer-aided environments.

The reasons for the use of SCM are both technical and contractual (Bershoff et al. 1980; Dart 1990). The technical aspects deal with the problems of software evolution and lack of control and understanding regarding the components, which create the configuration. Feiler (1991) further divides the technical SCM aspects into two disciplines, management and development support ones. In a management support discipline SCM deals with controlling changes to software, and in a development one SCM provides functions for assisting developers in performing co-ordinated changes to software. The previous discipline regards SCM as a support for project managers, whereas the latter one supports developers.

In addition to IEEE's standards IEEE 1042 (1987) and IEEE 610 (1990), some organisations, e.g., the Department of Defence (DoD) and ESA, have created certain standards for SCM, such as DoD-STD-2167A (DoD 1988) and PSS-01 (ESA 1989) which have directly affected SCM practices. The duty to follow standards has emphasised the role of SCM, creating a presumption for its development. In addition, SCM has also emerged as one of the issues in the quality-driven approach to software engineering (Sommerville 1995; Tervonen 1994).

According ***to the traditional definition*** of SCM given by (Bershoff et al. 1979) and also used by the IEEE 828 (1990) SCM includes the following basic elements:

- configuration identification - *identifying and defining* the configuration items for the product,
- change control including configuration control - *controlling* all changes to these items throughout the product's life-cycle,
- configuration status accounting - *recording* and *reporting* the status of configuration items and change requests, and
- configuration audit - *verifying* and *auditing* the completeness and correctness of these items.

The functional relations of the concepts are illustrated in Figures 5 and 6. This traditional SCM definition, given by Bershoff et al. (1979) and IEEE 610 (1990), has been extended by Tichy (1988) and Dart (1991) to include manufacturing issues, process management and teamwork. In this case software manufacturing means the process of generating derived configurations by a build mechanism. According to this ***extended definition***, SCM is a discipline whose goal is to control changes through the functions of: *component identification*, *change tracking*, *version selection* and *baselining*, *software manufacture*, and *management of simultaneous update*s. The manufacturing aspects are referred to as *release* production in the ESA software engineering standard (ESA, 1991) and *release management and delivery* in the ISO/IEC standard 12207 (1995).

When relating Feiler's (1991) classification to the previous SCM definitions, traditional definition sees SCM more as management discipline and extended one, in particular, including software manufacturing, as development discipline.

The basic concept of this SCM definition is that of a *Configuration Item* (CI), which can be a unit or a collection of lower-level items. CIs can also be regarded as specific instances of software products. As the terms are not well-established, some authors use the concept 'object' to describe CI.

Versions of CIs are understood to be either variants or revisions (Tichy 1988). More exactly Conradi and Westfechtel (1998) define a version to represent a

state of an evolving CI. Parallel *variants* of CIs are different implementations of the same CI, such as items executable on different platforms or based on different languages. An essential function from the point of view of variant manipulation is merging, because it is a means of collecting together the results of parallel versions. Serial *revisions* are CI steps that change with time.

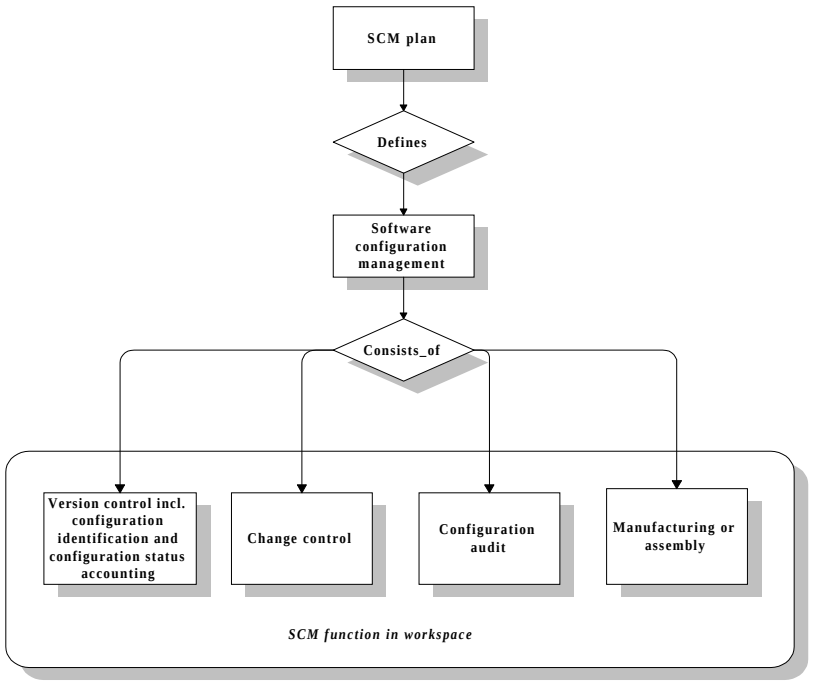

Figure 5. The basic elements of software configuration management.

Version control is an approach to the management of CIs stored in a repository or archive. Work environments are created by extracting these from the repository and conveying them to the workspace by a check-out command and vice versa by a check-in command[3]. One of its practical benefits is that it saves archive space, because versions of the same CI stored in the repository are based on their incremental changes rather than being full copies.

[3] The check-out/check-in commands are replaced by get/put commands in some version control systems.

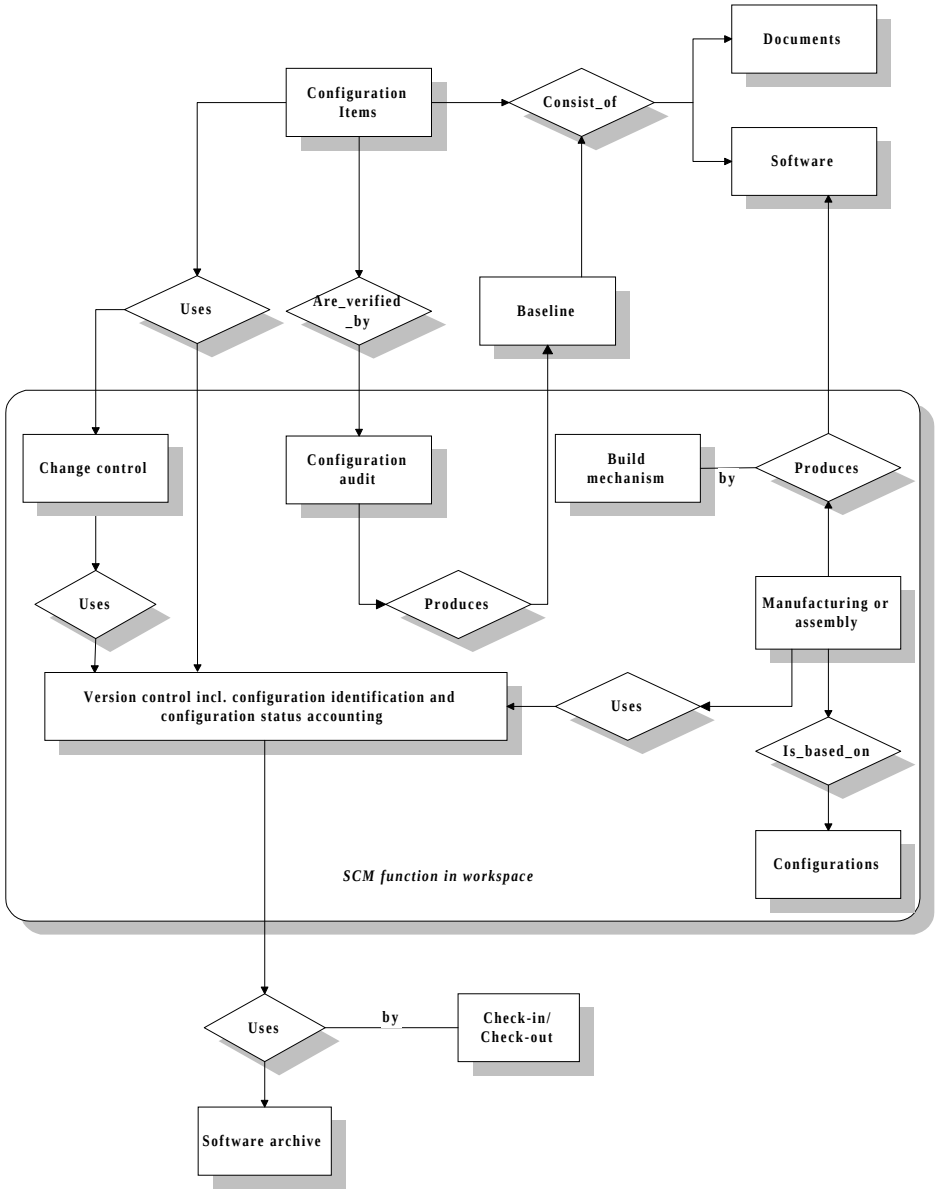

Figure 6. Software configuration management, concepts and procedures.

The classical version control systems are SCCS (Rochkind 1975) and RCS (Tichy 1985). They have gained successors in both the commercial and the research field (Dart 1991; Forte 1993; Rigg et al. 1995). The improvements to SCCS/RCS developed later concentrate on revision and variant treatment and

the incremental storage mechanism (Hunt et al. 1996). The differences between two versions are called *delta.*

The workspace, also called the work area, is used to manage copies of files retrieved from the repository. As a descriptive computer technology definition, a file in the workspace has been described as a cache copy (Tichy 1988; Feiler 1991). Workspace and repository solutions and their mutual relations can be used to characterise modern version control systems (Belkhatir et al. 1994). The solution classes range from check-out/check-in to integrated workspace-repository solutions embedded in commercial data management systems. A profound analysis of various workspace solutions is given by Conradi and Westfechtel (1998).

Software manufacturing is based on build mechanisms, which are implemented by a software assembly system. The terms are not yet well established. The ESA standard, for example, names this phase release management and delivery (ESA 1989 and 1991). A software assembly mechanism uses a configuration, which consists of CIs, linkages between CIs and compilation and linking commands stored in a configuration file. An example of an assembly system is discussed in Papers III and V. The system compares the time stamps of object and source files and infers the commands to be executed for assembling the software. An assembly system can speed up the process of producing delivery-specific software packages considerably. Examples of this principle are the traditional Make-based solutions (Feldman 1979) pioneered on UNIX systems and supported by the SCCS and RCS version control systems. A Make utility controls a build activity according to a configuration description, typically a script that defines dependencies between parts of assembled software packages. There are several corresponding tools available today, based on both public and commercial solutions (Gulla et al. 1993; Rigg et al. 1995; Conradi and Westfechtel 1998).

A baseline is a document or product that has been formally reviewed and agreed upon and can serve as a basis for further development. A baseline can be an assembly of CIs, an accepted configuration. A formal change control procedure is required for modifying baselines (IEEE 610-1990; IEEE 828-1990).

One of the baseline categories is a release. It is a particular version of a CI that is made available for a specific purpose (ISO/IEC 12207 1995). It is typically certain promotions of CIs that are distributed outside the development organisation (IEEE 1042 1987).

The managing of configurations and source code is essential but error-prone. Whenever the source code is modified, software developers have to check if some changes are required in the configuration. The programming-in-the-large philosophy presented by DeRemer and Kron (1976) extended the role of programming languages to higher abstraction levels. In this framework, Module Interconnection Languages (MILs) were developed as a solution to the interface problem (Prieto-Diaz and Neighbours 1986). MIL solutions can be regarded as an advanced software build mechanism, which makes it possible to link software modules implemented by different programming languages together dynamically. In a change situation there is a risk that interfaces may become inconsistent. MILs separate computation from connections between the modules that implement the computation. In maintenance it is possible that the connections are changed or the computations can be reused in other systems with different connections.

The major problem of one of the original MIL descriptions presented by Tichy (1988) is the difficulty of dealing with versions of interfaces, but there are results indicating possibilities for applying the MIL concept to SCM, ranging from programming language-based solutions such as Ada, CLU and Mesa to true MIL-based solutions (Tryggeseth et al. 1995). The MIL language Polylith developed by Purtilo (1994) was applied by the author to design descriptions of embedded software, and the relationships between embedded software and specific MIL descriptions were evaluated and tested using this MIL language (Taramaa and Purtilo 1990).

In parallel, the integration of different Computer-Aided Design (CAD) environments for electrical and mechanical parts has been supported by creating the STEP (STandard for the Exchange of Product data) standard (ISO 10303-1 1994). One of the goals of this standard is to facilitate interoperability when there is need to exchange standardised data product models. For the description of data models there is a specific language EXPRESS (ISO 10303-11 1994)

which can be regarded as an implementation of interconnection of the CAD descriptions (Estublier et al. 1998).

Some of the 'second generation' SCM tools, such as Cedar (Swinehart et al. 1986) and Gandalf (Habermann and Notkin 1986), also employed the MIL approach (Ambriola et al. 1990), but they did not gain popularity, since the industry was not ready to make use of such integrated, comprehensive software engineering solutions in the mid-1980s, when the general trend was to develop a software engineering environment that included solutions for all the software engineering activities. Some MIL-based software engineering environments have been developed, however. Sommerville and Dean (1996) give a comprehensive overview of existing MIL languages, comparing these with the capabilities of the MIL language PCL produced by the ESPRIT project PROTEUS.

Interconnection presentations are nowadays classified more generally as configuration programming (CP) languages, an illustrative survey and comparison of which is given by Bishop (1994). The potential target area of CP languages is dynamic configuration of large distributed applications, where they have achieved some success (Thornton 1996).

Modern commercial SCM environments have also borrowed from configuration solutions. Adele, for example, creates a configuration model before its realisation (Estublier and Casallas 1994).

The above discussion is concerned with SCM procedures. In a more comprehensive context, the SCM plan is a framework for instantiating a certain SCM solution for a company. It defines each SCM element, with procedures and their practical implementations. In the new ISO/IEC 12207 standard (1995) SCM planning is taken as an equal SCM element to those mentioned in the extended definition, called process implementation. More detailed guidance is given by the IEEE 1042 (1987).

SCM solutions including the SCM elements described above are displayed in Table 4. The SCM elements of the extended definition are extensions to traditional definitions, while the comprehensive context elements are extensions to extended definitions.

Table 4: A summary of SCM elements.

SCM definition categories	SCM elements	Description
Traditional definition	Version control including configuration identification and status accounting	Solutions for configuration identification and configuration status accounting
	Configuration audit	Verification and validation mechanisms
	Change control	Configuration control throughout the product's life cycle
Extended definition	Software manufacturing based on conventional builders	Traditional compiling and linking technique
	Software manufacturing based on configuration languages	Dynamic linking technique
	Teamwork	Communication principles and mechanisms
Comprehensive context	SCM planning	SCM described as a part of the defined processes

2.2.2 Configuration management of embedded software

In the area of embedded software systems, the very first computerised SCM solutions were adopted by the manufacturers of telecommunications and aerospace systems (Berlack, 1992). These applications were so large that SCM was not possible without computerised tools.

Since embedded computer systems are collections of tightly coupled hardware and software components, efficient communication between the respective representatives is necessary in configuration auditing, e.g. for defining interfaces between hardware and software. Examples of these interfaces are automation design related to software design in mechatronics applications and electronics design related to software design in applications including digital signal processing.

A relatively long delay in hardware availability for software development is also typical of embedded applications, since parallel hardware development, such as ASICs, might take much more time than software development (Mittag 1996). These features have resulted in specific solutions for the development of embedded software. Prototyping has proved beneficial (Agresti 1986; Boehm 1988; Pulli 1991), but it places special requirements on SCM, since the functions implemented by the software evolve through prototypes (Niemelä et al. 1995; Seppänen et al. 1995, Savola et al. 1995). In practice the special requirements appear as specific CIs, such as special CASE descriptions, combination of completed code and specification test results of various heterogeneous prototypes. In addition to specific CIs, prototyping of embedded software includes a strong concurrent approach (Pulli and Heikkinen 1993; Heikkinen 1997). Software itself is developed in parallel with other technologies, and for this reason efficient management of the various CIs needs to make use of advanced SCM solutions, such as strong team co-ordination, naming disciplines, minimising change risks, strong support of design history.

Safety and security requirements of embedded software need to be taken into account. The analysis of these requirements demands the usage of specific testing and simulation tools (Honka 1990; Latvakoski 1997). The results of testing and simulation are input for configuration audit where the critical parts of software are reviewed by a nominated board. The critical requirements are defined by test specification, which acts as a reference to testing and simulation.

The software supply business is becoming increasingly focused on reuse (Davis and Bershoff 1991; Gomaa 1993; Seppänen et al. 1993; Frakes and Isoda 1994), SCM implies a need to record and control software configurations for different projects and customers. The requirements for SCM are caused by the nature of embedded systems, where the provision of suitable software calls for a strong software manufacturing mechanism (Kemppainen 1986; Östlund and Forssander 1996). Implementation of the reusability mechanisms is one of the most essential elements when fulfilling this requirement.

The special nature of software manufacturing can be seen in two aspects: specific hardware-related CIs and the final result, i.e. the image, which will be duplicated on several chips. Some examples of software assembly alternatives originally presented in Paper III are given in Table 5.

Table 5: A list of potential assembly alternatives.

CONFIGURATION STRUCTURE	**CONTENTS**
APPLICATION SOFTWARE	the highest level assembly consisting of the binary to be embedded, reporting binaries, design descriptions, and other documents
DESIGN DESCRIPTIONS	to be linked to the delivered software product
BINARY EMBEDDED SOFTWARE	an assembly to be created by copying different binaries, such as application tasks, real- time operating system modules, communication software, and initialisations
APPLICATION TASKS ...	an assembly using a makefile consisting of application tasks, communication software, real-time operating system, and hardware modules
DOCUMENTS	e.g. the usage guides of a product, to be linked to the delivered software
INITIALISING DEFINITIONS SOURCE MODULES OBJECT MODULES	an assembly using the conventional initialising activities; similar to application tasks, but consists of specific initialising functions as well as hardware modules
REPORTING	an assembly using a makefile consisting of of application tasks and the operating system for non-real-time tasks
COMMUNICATION SOFTWARE INCLUDE MODULES OBJECT MODULES BINARY MODULES	 an assembly using the makefile
HARDWARE DESCRIPTION MODULES BINARY MODULES	 commercial binaries
REAL TIME OPERATING SYSTEM DEFINITIONS BINARY MODULES COMMANDS	commercial code ranging from source code to binaries
OPERATING SYSTEM FOR NON REAL TIME TASKS	
BINARY MODULES LIBRARY MODULES INCLUDE MODULES	commercial code ranging from source code to binaries

On the highest level there is the application itself, assembled from various binaries such as application tasks, real-time operating system modules, communication software and initialisation routines. The makefile at this level only includes a copy operation performed in a specific order. The creating of application processing modules with real-time features involves more typical makefile features, such as the activation of compilers and linkers.

The close relation between software and hardware technologies is placing SCM to a new situation. Developments in integrated circuit technology are altering the traditional software development practices, since the work can now be also based on system-on-silicon design solutions where the idea is to replace a computer system with a chip (Heusala and Tiensyrjä 1995). In this situation an efficient co-design including, e.g., control software, digital signal processing software and VHDL descriptions producing ASICs and FPGAs, makes CM into a versatile procedure with various tools to be linked to it. In addition, product development is then closely directed by the silicon implementation architecture and the software modules become equal to other more hardware-oriented modules managed by the CM system.

The above description applies to electronics products, but the perspective can be extended. The customer order for a product based on an embedded system will include several technologies to be specified, designed and implemented and will result in the assembly of mechanical as well as hardware components and the embedded software. In addition, other product technologies and their design descriptions, e.g. automation design descriptions of Paper III, are extensions to configuration description. The relationships between the entities needed for the production of embedded computer systems according to a customer order are illustrated in Figure 7.

The trend for a movement towards the integration of different technologies, including software, among others emphasises product-level information processing. Product Data Management (PDM) systems are an example of this development. CM is seen as a link for maintaining consistency between parts implemented by various technologies, documentation and change data, from design to manufacture and support. The developing of a comprehensive CM system requires an understanding of the relations between the different technologies. Dart (1992b) provides a good comparison between software and hardware concepts in relation to PDM-based CM.

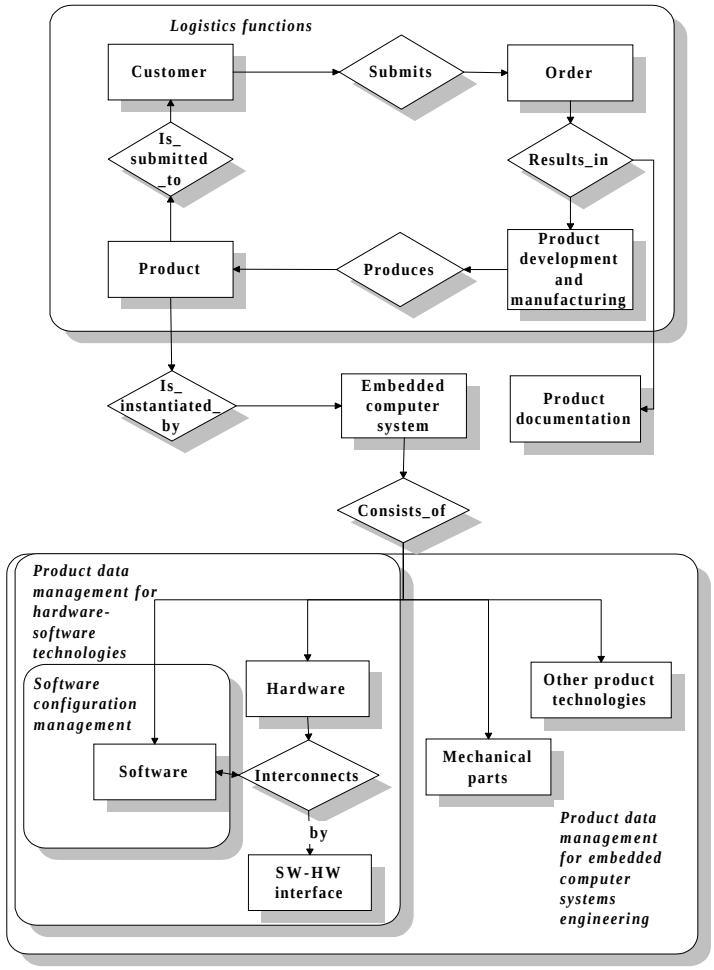

Figure 7. An order resulting in a product and product documentation.

PDM systems originated from hardware CM systems during the 1990s through the addition of SCM (Sherpa 1995). This is an important extension, particularly when companies are developing integrated company-wide information systems for different computer-based functions (Tiihonen et al. 1995). A good example of the integration of several product development technologies is given by Deitz (1990).

2.3 Change related activities

Control over changes is one of the basic SCM functions. Changes in the context of software maintenance have most often been examined from an application-independent viewpoint, because general evolution patterns can be found in all software applications. The same approach is taken here, but extended to embedded software applications.

2.3.1 Maintenance in software engineering

The importance of maintenance was underestimated until the late 1980s. Foster et al. (1989), outlining the general attitude to software maintenance prevailing in the 1970s and 1980s, note that it was seen simply as an activity carried out at the end of the software development process or as a series of feedback lines implying repetitions of certain activities, or even as a software engineering task that could be completely ignored. This is indicated well by the first life-cycle models, in which maintenance is seen as an activity performed only after the development process. The IEEE software maintenance standard IEEE 1219 (1992) enforces this principle, stating that software maintenance is the ”modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a changed environment”. This implies that changes made before delivery of the software can be regarded as software evolution.

The general categories of adaptive, corrective, perfective and preventive software maintenance were proposed in the late 1970s (Swanson 1976). In adaptive maintenance the software is enhanced to adapt to environmental changes, while corrective maintenance concentrates on diagnosing and correcting errors, perfective maintenance like its adaptive counterpart, enhances the software, but by altering its functionality, and preventive maintenance considers the needs for software maintenance over a long period of time and updates the software to anticipate future problems.

The work discussed by Belady and Lehman (1976) and Lehman (1980) can be regarded as providing the foundation for a more comprehensive approach to software maintenance in which changes are addressed throughout the software life-cycle.

Many experiences have proved the usefulness of this point of view. Software maintenance has been shown to account for between 40% and 70% of all software expenditure (Foster 1993; Krogstie and Sölvberg 1994). In the classical analysis by Lientz and Swanson (1980), 17% of all maintenance was corrective, the rest being mainly perfective and adaptive. Corrective maintenance can be seen in this taxonomy as ”pure” maintenance, while the other types include greater or lesser measures of evolution. Later results have been similar, with corrective maintenance typically amounting to about 20% of all maintenance activities (Krogstie and Sölvberg 1994).

ESA has further extended the maintenance categories (Harjani and Queille 1992; Stefanelli and Stroobants 1991). User support answers explicit requests for information from the user or corrects his misunderstandings, evolutive maintenance evolves and/or expands the needs of the users by means of new functions and anticipative maintenance is directed at problems with the use of the software or with the evolution of its operational environment.

These three definitions include some overlapping with two of the four discussed earlier, so that the categories of perfective and preventive maintenance need more exact definitions. In this new classification preventive maintenance aims at improving the maintainability of the software and perfective maintenance at improving its non-functional properties. This is a more natural role for perfective maintenance, since the changes and additions of new functions are regarded as system and software evolution rather than maintenance. The benefits of these categories can be observed when developing the maintenance process by linking specific tools to it. The differences in nature between the maintenance categories provide an opportunity to utilise this aspect in process models.

2.3.2 Maintenance of embedded software

As implied by the above discussion, changes can involve several parts of the embedded software: the operating system, communication and device control (special system software), and common or application-specific parts of products. In addition, changes in hardware requirements, specifications and design solutions usually have some impact on the software, and vice versa.

The nature of maintenance in industrial organisations, in terms of the categories perfective, adaptive, corrective and preventive, is analysed in Paper I, where it is shown that 75% of the organisations performed perfective, adaptive and corrective maintenance.

Each software and hardware component may have versions independently or as a consequence of a modification made to another component. For example, if an input/output controller chip is no longer available, the hardware design of a sensor interface in an embedded system based on that chip may need to be modified. A new version of the device driver program used to control the interface would then have to be developed, although the rest of the software may not require any modification. The resulting effect is that functionally similar embedded software packages may include different components.

Changes to the control software of an embedded system after delivery to the customer may be made according to different maintenance policies. If the electrical board of a system is damaged, for example, the manufacturer of the product may respond to the maintenance request either by replacing the damaged board and its original control software package with a new copy, or by supplying a new board that includes new device driver programs but the same application software as in the damaged board.

Some changes may be made as part of corrective maintenance initiated by the manufacturer of the product. If a software error has been found in one or more items supplied, for example, a corrected version of the software may have to be distributed to customers. Other changes may involve additional system features required by a specific customer, so that a new software package may have to be developed for that customer.

The integration of separate software products related to embedded software has speeded up demands for better management of software maintenance. With the increased numbers of interfaces between software products, maintenance has become even more demanding. This has been emphasised especially with embedded software, such as BIOS (Basic Input/Output System) software, with that used as a part of PC applications serving as a good example of this situation (Nummelin 1998). Although the PC hardware develops very fast, the BIOS software has to look similar to applications.

Maintainability in embedded software can be achieved by various means. In addition to change management and CM environment aspects, the target software to be maintained is important (Oman et al. 1992). The connections with the hardware make the maintenance of embedded software different from that of non-embedded software. Embedded software can be developed from the point of view of potential changes and enhancements (Adamson 1990), the practices recommended being the use of relocatable code and data whenever possible. The constants recorded in the ROM can be grouped together, and the code itself uses these constants indirectly by reading them from a table using pointers. Text strings are stored together, offering either a choice of languages or the possibility of replacing one block of text with another.

In embedded systems the software often represents the result of an evolution-oriented development process, including features of the different maintenance categories, the resulting software, or image, is transferred to the electronics manufacturer as part of the custom-made mask. This makes the economical production of ROM chips by means of long production runs feasible, but it also poses challenges for evolution-oriented development, since the software cannot easily be changed later.

In some situations patching is the only way to modify software after chip manufacture. Dynamic program-updating systems can make it possible to repair bugs and enhance running software without the cost of system shutdown (Segal and Frieder 1993). Typical examples are the solutions used in on-board space software (Alonso and de la Puente 1993; Lopéz and Rodríguez 1995) and telecommunications switching systems (Frieder et al. 1989; Hauptmann and Wasel 1996). This approach is not yet recommended, but in some cases it is the only feasible way to maintain software.

Programmable Read Only Memories (PROM) provide opportunities for a different production cycle, because they do not need to be programmed until they are added to the circuit board (Rosch 1994). This also places software maintenance in a slightly different situation, because the software is not totally connected with electronics manufacturing. On the other hand, although evolutive software development is separate from electronics manufacturing, there is a need to recognise the fact that the software cannot be changed after storage in the memory.

Erasable PROMs (EPROMs) enable the same program to be produced several times by erasing and reusing the program memory. The role of the software can be regarded as comparable to that in traditional data processing applications with a fixed processor platform. Flash memories have placed embedded computer systems in a new situation because of their more flexible erasing and reprogramming features compared with previous erasable memory solutions (Grundmann 1997). The connection with electronics manufacturing technology is not a firm one, although the change effects between hardware and software disclose the basic features of the embedded system and its software, i.e. the close connection with the hardware.

2.4 Process modelling for maintenance SCM

All software is produced by some process, but it is often an incoherent and implicit one. When defining more explicitly processes the central description elements of processes are activities to be done, artefacts to be produced and used by a process, tools to be used to implement the process, roles to be defined by responsibilities and rights, and agents to be related to roles (Conradi et al. 1994).

The development of software process technologies, i.e. methods and tools for modelling and enacting software processes, emerged in the mid-1980s. One of the best-known articles on this subject is that of Osterweil (1987). The spiral model proposed by Boehm (1988) is also regarded as a blueprint for modern process modelling approaches.

Process models are also used to describe the results of software process. When related to various software processes, the comprehensive software process model acts as a unifying element indicating how specific subprocesses are related to each other, e.g. SCM to software development and maintenance. Models can be categorised according to their nature, ranging from descriptive and prescriptive to enacted ones. Enacting a software process model, which is the target of active research and development, enables software engineers to carry out the process according to the activities specified in its model. Process improvement can be seen as a yet higher category in this taxonomy (Madhavji 1991; Christie 1994).

The plan of a specific process, e.g. the SCM plan, depicts the implementation of a process. Thus the plans are related to processes and are used as tools for management, since they include definitions of activities and procedures, schedules for performing these activities, organisational responsibilities and relationships with other processes. The relationships between these process concepts are described in Figure 8.

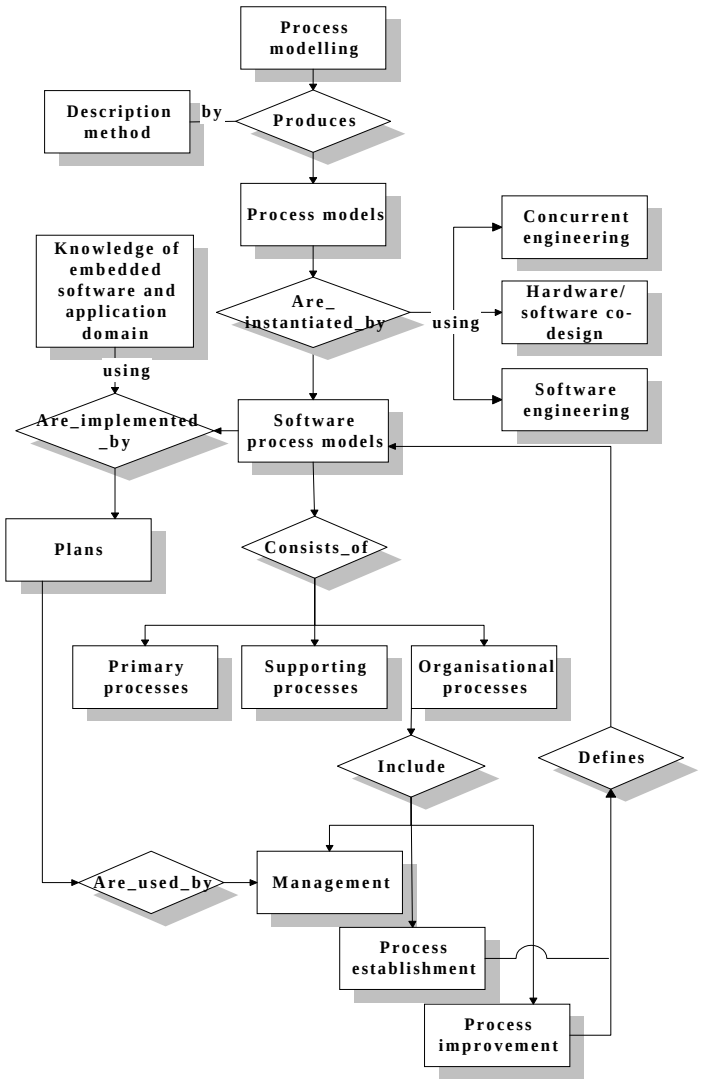

Figure 8. Process concepts related to software engineering, its processes and knowledge of embedded software.

The reference model of Figure 8 is based on the ISO/IEC 12207 standard (1995), which categorises software processes into three major groups: primary, supporting and organisational processes. Knowledge concerning embedded software and its specific features in particular presented in sections 2.1 and 2.2 is related to software processes as an input to the implementation of plans. The first category, primary processes, includes primary life-cycle process, e.g. development, operation and maintenance. The second category, supporting processes, consists of supporting life-cycle processes, e.g. SCM and quality assurance. The last category, organisational processes, is used to establish and improve the previous processes by providing sufficient resources such as process establishment and management.

The SCM concepts presented at the beginning of this chapter, the plans produced by the process model approach and software maintenance are further related to each other in Figure 9. The following sections further analyse SCM and related matters separately. Since the software maintenance process produces the extensions for change control within SCM, the maintenance process will be described in more detail. Software maintenance and SCM can be seen as a part of a more comprehensive framework, called application management, which it makes possible to take into account the effects of process establishment as mediated via plans.

2.4.1 Software maintenance process

Several researchers have proposed models for the software maintenance process (Bennett et al. 1991). One of the principal models presented in the mid-1970s includes three main phases, understanding and modification of the existing software and revalidation of the modified software (Boehm 1976).

As software engineering has become an industrialised process, request-driven approaches have come to dominate the process of software maintenance. These approaches recognise as the main phases those of request control, change control and release control.

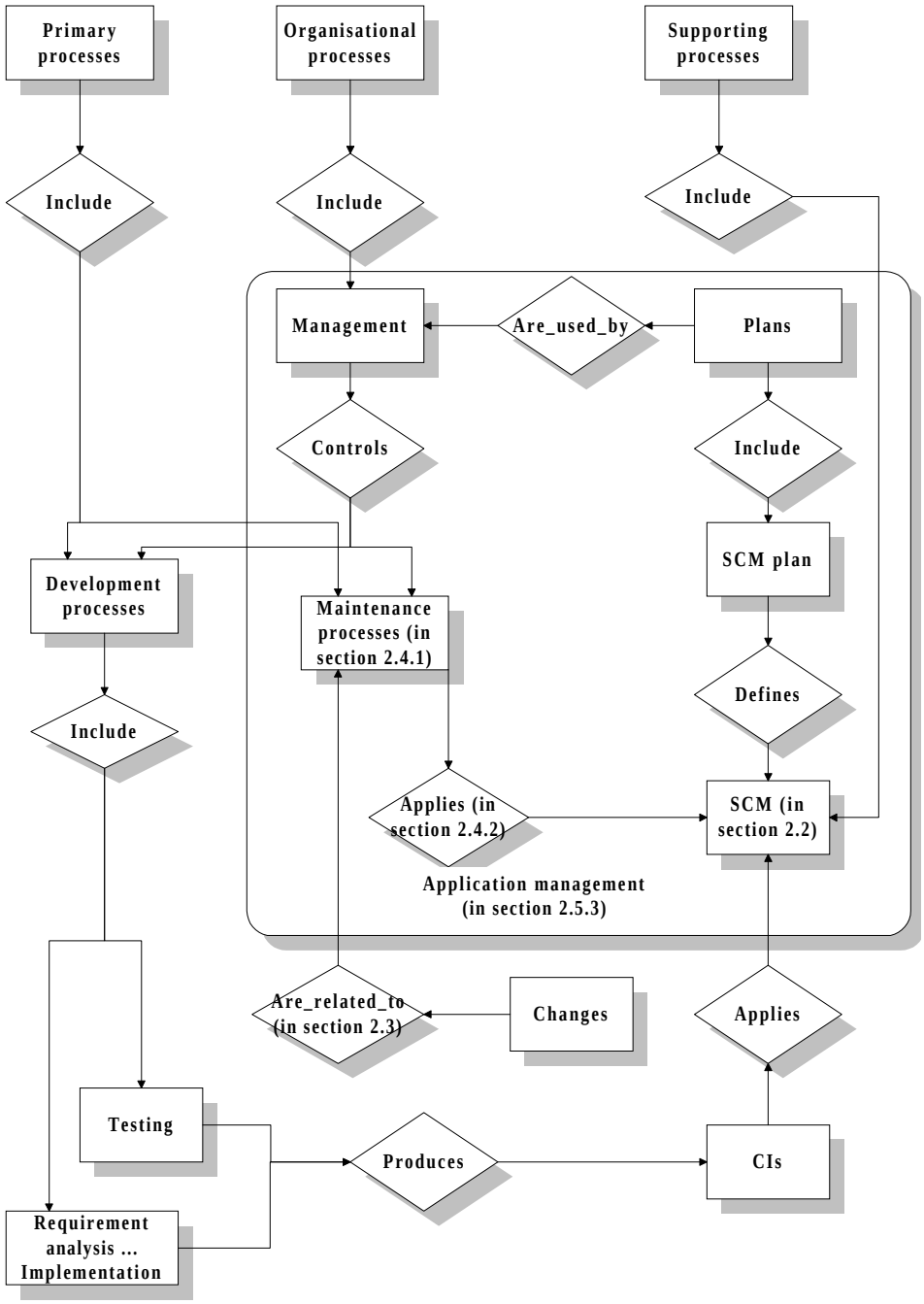

Figure 9. Software processes related to SCM.

The international standards, e.g. IEEE 1219 (1993) and ISO/IEC 12207 (1995), follow the same main structure. In addition to modification implementation, ISO/IEC defines the maintenance process as including problem and modification analysis, maintenance review/acceptance, migration and software retirement. The ISO/IEC 12207 definition includes the same maintenance phases as the previous definitions but is extended to cover the whole life-cycle until retirement and to allow for more comprehensive integration.

A more detailed model for the maintenance process based on the above division, as provided by the European Software Factory project ESF/EPSOM, is presented in Figure 10 (Harjani and Queille 1992). This consists of change control (the left side), implementation of changes, i.e. software manufacturing (the lowermost point of the model), and testing of the results (the right side). It can be regarded as an application of the request-driven approach, defining specific subprocesses for software maintenance.

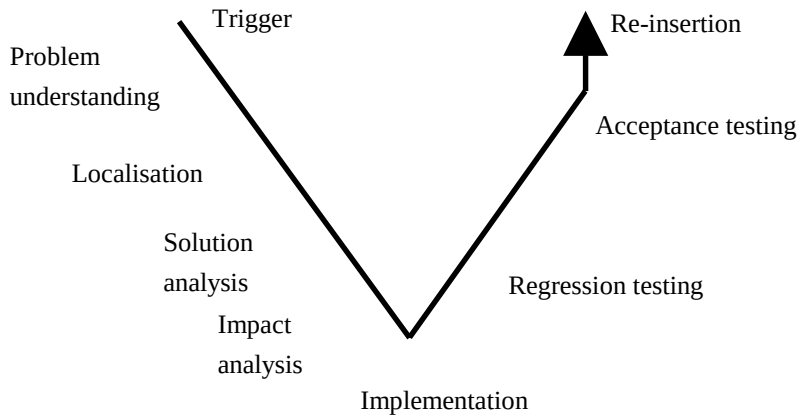

Figure 10. Software maintenance process.

A typical trigger for the maintenance process involves information about a software problem or a request for change. The steps in the process, also called pre-implementation activities, include problem understanding, localisation, solution analysis and impact analysis. Problem understanding is focused on the need to address the change, localisation concentrates on the target of the change, solution analysis deals with one or more changes that are devised, and finally

impact analysis evaluates the consequences of the changes before implementation.

These subprocesses can be supported by specific tools, examples of which are presented in the following list. Most of them were produced in the ESPRIT project AMES, which was focused on application management (AMES 1995):

- a hypertext-based application understanding (AU) toolset (Boldyreff et al. 1995; Laitinen 1995 and Laitinen et al. 1997; Vierimaa 1996; Paper VII) and reverse engineering (ReverseNICE) tools (Battaglia and Savoia 1997; Paper II) for problem understanding,
- a traceability platform for a navigation and display tool (Doize et al. 1994) to assist localisation and solution analysis, and
- a tool for impact analysis (IAS) (Barros et al. 1995).

The AMES project included an experiment with an enacted process, which related the tools listed above by means of a commercial enactment tool (Taramaa and Ketola 1995). This experiment applied processes originally presented by Stefanelli and Stroobants (1991) and pointed to several common functions such as solution and impact analysis, implementation decision and modification implementation. The beginning of the maintenance process includes specific instantiations of problem understanding and localisation, depending on the type of change request concerned (Figure 11).

The recent definitions of software life-cycle processes single out localisation, also called software resolution (ISO/IEC 12207 1995), problem resolution (ISO/IEC 15504 1995) or problem response system (Bell 1994), as a distinct support process, which includes technical solutions made by problem tracking systems. Problem tracking (also known as change, defect or bug tracking) refers to the process of recording and tracking change requests and deciding which changes to make to a software system (Chapin 1989). Problem tracking can be implemented at different levels, such as error database, linkage between error database and source files, construction of linkage between an error database and versioned source files via a version control system, or the use of an error database for further analysis and process improvement.

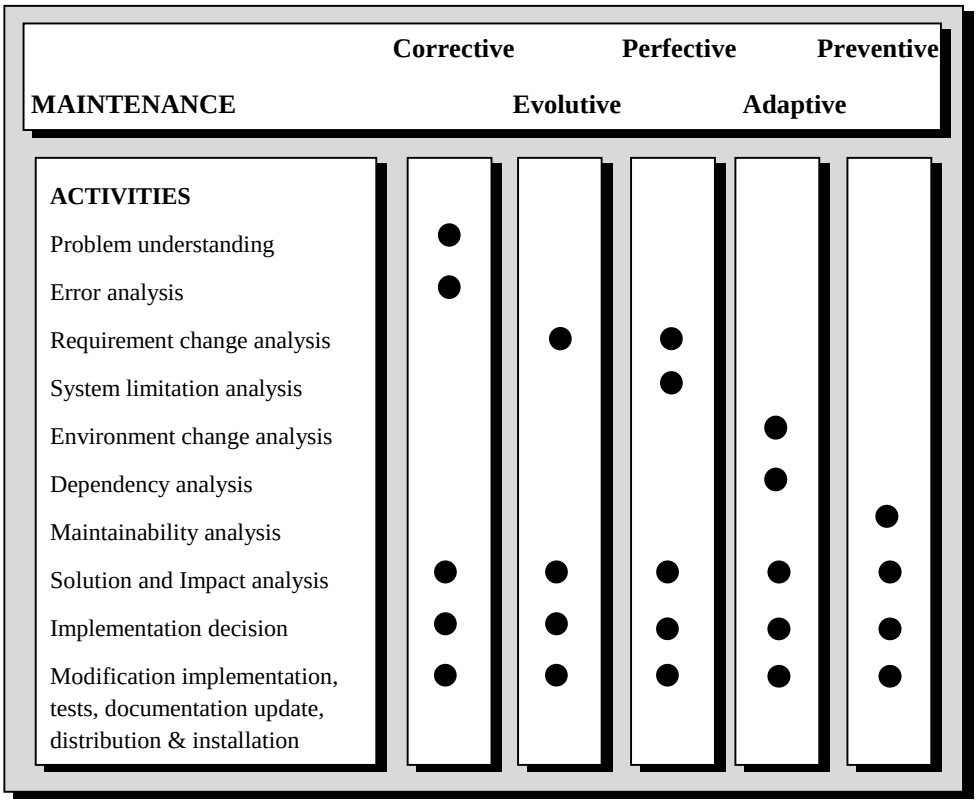

Figure 11. An example of the software maintenance process and its subprocesses based on maintenance categories (Stefanelli and Stroobants 1991).

Problem reporting can be linked to all of these systems, ranging from a simple error database to an advanced environment and including support for analysis and process improvement. Process-oriented support of problem reporting has already been developed in order to allow fast reporting of faults to the relevant parties (Juopperi et al. 1995).

A corresponding approach to that of ESF/EPSOM has been developed by Capretz and Munro (1992), who propose four software maintenance categories: perfective, adaptive, corrective and preventive. By defining the maintenance model, the phases through which a change should proceed during maintenance are clearly defined.

Preventive maintenance can be implemented by means of reverse engineering tools, for example, a situation described by Chikofsky and Cross (1990) and for the case of embedded software by Paper II and Rytilä (1994).

2.4.2 Software maintenance versus SCM

When considering software maintenance, change control creates a link with SCM, because change control of the SCM system implements functions required by pre-implementation activities. Change control or change management is a specific subject that is often held to be conceptually distinct from the other SCM activities, but in the most advanced SCM environments it is included as an inseparable part of the comprehensive SCM scheme. Change management provides a process for evaluating, co-ordinating, approving (or rejecting) and implementing changes to an established baseline version.

The relation between the maintenance process and SCM is depicted in Figure 12. Software maintenance requires change control solutions for specific pre-implementation activities such as problem understanding, localisation, solution analysis and impact analysis.

Based on the experiences obtained in the development, embedded systems-specific features in use of these tools can be analysed as follows:

- Problem understanding: Application understanding toolset includes the definition of domain-specific concepts where the structure of embedded systems and software has to describe for making useful links to hypertext descriptions. Reverse engineering tools are especially focused on real-time software producing descriptions for HOOD and RT-SA/SD.
- Localisation and solution analysis: Navigation and display tool is general. It is C language-specific.
- Impact analysis tool: Tool is C language-specific.

Taramaa and Ketola (1995) recount their first experiences of the instantiation of SCM implemented by a process enactment tool, and the subject is dealt with more thoroughly in an application-oriented pilot project where the central part of the system is a problem tracking mechanism and the target the maintenance of on-board software (Harjani et al. 1995b).

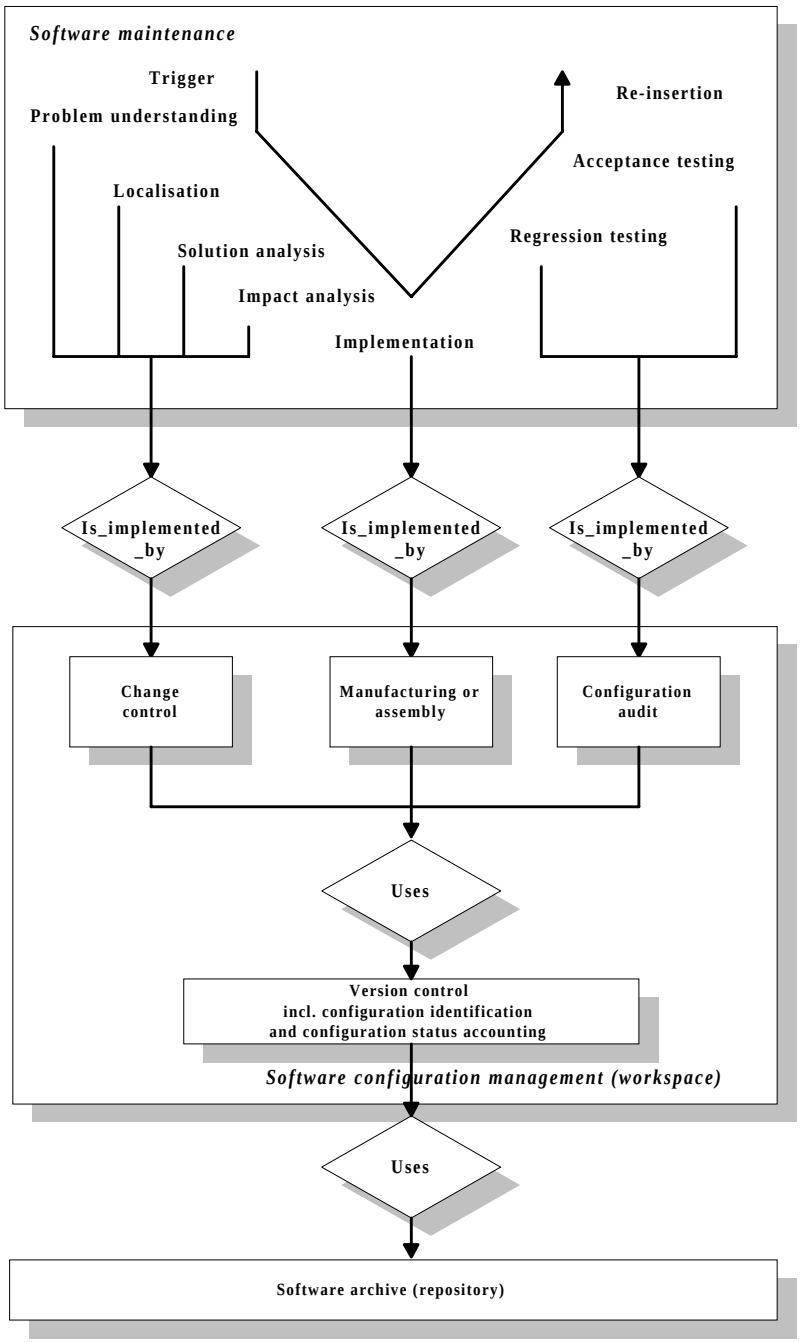

Figure 12. Relationships between the software maintenance process and SCM.

2.4.3 Extended software maintenance process - application management

Since the request-driven model does not stress different levels of abstraction in the maintenance process, extensions have been developed such as the ESPRIT project REDO (Bennett 1993), emphasising the business view.

The maintenance activity requires specific solutions for higher levels of abstraction in management. The definition of application management proposed by AMES (1995) is as follows:

> *"Application management is the contracted responsibility for the management and execution of all activities related to the maintenance and evolution of existing applications, with well-defined service levels".*

The maintenance process model developed by the AMES project (AMES 1994a) is divided into three levels that indicate the need to differentiate the roles of people participating in the maintenance process (Figure 13).

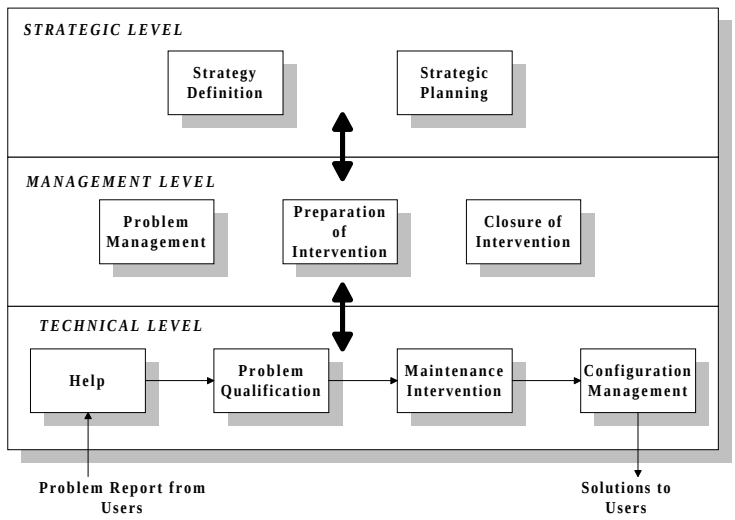

Figure 13. The AMES model for application management (AMES 1994a).

The strategic level emphasises that companies have their own strategies concerning software maintenance. When relating to the software processes, this level is an instantiation of process establishment, referred here as strategy definition. The management level stresses control activity. In addition, it is the responsibility of a project manager to transfer information to and from each of the other levels. From the point of view of the software processes, this level corresponds to the management of organisational processes. The technical level of the application management involves implementation of the software maintenance process, including SCM functions. Figure 12 can be regarded as a more detailed description of the technical level.

Accordingly, it follows the ESF/EPSOM request-driven model and is associated with REDO's higher level business aspects, its strategic and management level. A profound analysis and comparison with related approaches is given by Boldyreff et al. (1994).

The application management model can be seen as a framework that must be instantiated in each industrial environment. The example provided by Taramaa and Ketola (1995) is of an instantiation of this process concerning ESA software maintenance practices, based on the ESA software engineering environment project ESSDE (Aumaitre et al. 1993; Coene 1992; Favaro et al. 1994).

2.5 Other processes

Like problem resolution, software documentation has been typically regarded as a distinct support process, referred to as documentation (ISO/IEC 12207 1995) or develop documentation (ISO/IEC 15504 1995) or internal/user documentation (Bell 1994).

From the SCM viewpoint, it is essential to keep documentation consistent with other configuration items. In particular, the link with configuration presupposes valid documentary information on various development and maintenance situations.

In addition to the engineering and its support processes, there are logistics processes, such as purchasing, manufacturing/production and sales/marketing

processes that have to be taken into consideration (Figure 14). The engineering processes cover several computer-aided activities, such as Computer-Aided Engineering (CAE) and Computer-Aided Design (CAD) of several product technologies, which are used in product development of embedded systems. Manufacturing/production includes Computer-Aided Manufacturing (CAM) solutions. In addition, the Material Requirement Planning (MRP) systems based on the computerised Bill-of-Material (BOM) systems including product structures have acted traditionally as a common framework for managing and co-ordinating product data between manufacturing/production, sales/marketing and purchasing processes. (CIMdata 1996)

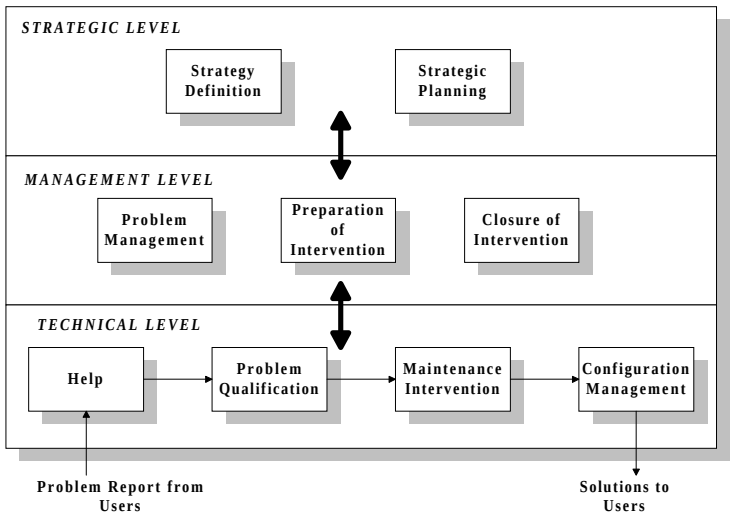

Figure 14. Enterprise processes related to product data management.

The current trend is still increasing integration of these processes. In particular, integration of CAE/CAD/CAM activities with MRP is making Product Data Management (PDM) as more viable in industry. The logistics processes also include data items, which have to relate to the software life-cycle processes. In addition to product data, the data items common to these logistics processes include customer data and delivery data. (Sherpa 1995).

The software development from the point of view of SCM has been typically left undefined at the product level, although embedded software forms one of the technologies of the comprehensive PDM. The main targets to be developed at this level are all product-related issues as opposed to software only that the Figure 15 shows by relating software and SCM to the whole product configuration and PDM.

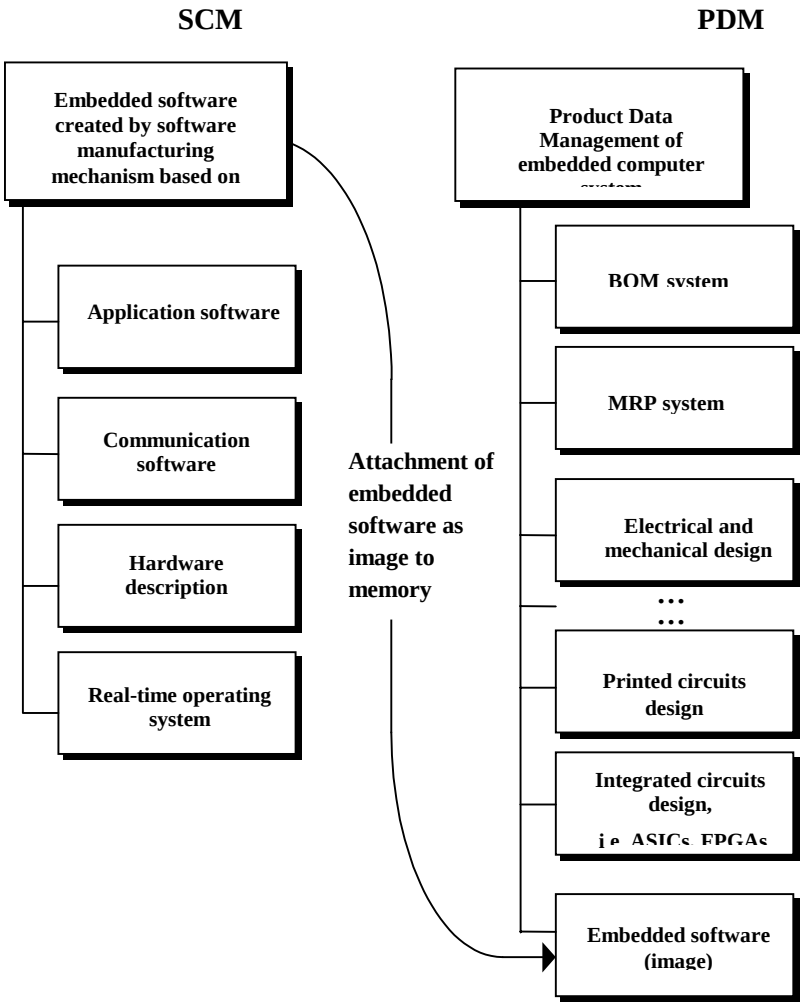

Figure 15. SCM related to PDM.

2.6 Related work

The best examples of the approaches to SCM practices are provided by Feiler and Dart. Feiler (1991) views SCM as transactions with the SCM workspace, while Dart (1990) presents a set of factors related SCM which have to be assessed when taking automated SCM systems into use, adopting an approach which concentrates on situations in which companies are automating their SCM systems.

The modelling-based approaches act as a unifying factor for different software processes. In space applications much effort has been invested in process development projects to model the software engineering practices to be used, and SCM has been one of the processes developed. Since SCM is related to other software process activities, this application area provides a practical view of advanced SCM solutions.

Assessment-based software improvement approaches provide a comprehensive view of the software process, i.e. an opportunity to improve both the software process as a whole and its specific subprocesses, including SCM.

The related approaches are evaluated below with regard to the framework created in this research, including SCM concepts, elements and levels, relating the elements to the levels and considering the improvement of SCM practices.

2.6.1 Approaches to SCM practices

At the beginning of the nineties, both Feiler and Dart published a profound analysis concerning SCM and problems in its use.

Feiler's SCM models

The starting point of Feiler's SCM taxonomy (Feiler and Downey 1990; Feiler 1991) is to separate SCM basic elements and to see different levels to implement them. Feiler has described SCM by means of four models:

- *Check-out/Check-in model*, which provides version control of individual system components,

- *Composition model*, which focuses on improving the construction of system configurations through selection of alternative versions,
- *Long transaction model*, which emphasises the evolution of systems as a series of configuration versions and the co-ordination of concurrent team activity, and
- *Change set model*, which promotes a view of configuration management focused on logical changes.

The check-out/check-in model is known as a traditional SCM model. It provides support for versioning of individual components and concurrency control by a locking mechanism. Feiler notices that this version control with manual repository manipulation is one the models to be applied, for example, by the first commercial tools. The composition model includes two main steps, i.e. the definition of configuration structure and qualification of the right versions of each component. This model is focused on software manufacturing with advanced building mechanisms. The long transaction model emphasises the role of workspace which consists of a working configuration and a series of preserved configurations. Creating specific workspaces for separate developers or development teams can be regarded as a way to manage concurrency. The change set model emphasises support for managing changes throughout the life cycle of each product family software.

Dart's adoption model for SCM

The model proposed by Dart (1995) is focused on the idea that SCM systems have been developed starting out from the in-house SCM systems with manual procedures and policies originally used in the past. The presentation provides a better understanding of SCM, a common vocabulary for it and commercial tools, while the future presents challenges for the further development of SCM, including specific viewpoints such as technical, process-oriented, political standardisation and managerial aspects (Dart 1992a).

Based on this vision, Dart has further developed and extended the framework presented in 1992. She sees this framework as an extension of a more general technology transfer approach, and names the viewpoints to be analysed as technical, managerial, process-related, organisational, cultural, political,

people-related and risk-related. She also analyses the risk-related aspects (Dart 1996).

The similarity between Dart's work and the present research lies in the incremental adoption of new SCM features. Dart concentrates comprehensively on a number of factors, which have to be taken into account in SCM improvement. The position of technical issues is not emphasised, but she appreciates the difficulties attached to current software development environments, particularly large ones. Small technical steps demand comprehensive analysis of the software development environment.

2.6.2 European Space Agency's (ESA) process models

The starting point for the ESA life cycle specification is its PSS-05 standards (ESA 1989 and 1991). The situation can be regarded as analogous to that of the DoD-STD-2167A and ISO9001 standards. ESA's software engineering practices have been developed further in the ESSDE projects (Favaro et al. 1991; Aumaitre et al. 1993), including specific solutions for software maintenance (Stefanelli and Stroobants 1991) and SCM (Aumaitre et al. 1993).

The ESA/ESTEC project PROMESSE, as described by Harjani (1993), has produced process models that include entity-relationship definitions for SCM and change control, and also their process descriptions. The AMES project (AMES 1995) has further developed the process model for software maintenance, starting from the concept of software change request (SRD) (Harjani et al. 1995a). An example of this model is presented in Paper IV.

The role of the ESA projects is slightly different from that of assessment projects. ESA's goal is to fix all software engineering functions, since most of its projects are based on collaboration. To define a common process model for all functions is seen as a way of improving software development practices in its projects in general. SCM practices form one of the most significant elements used to define this collaboration.

The ESA approach provides an example of how to develop software engineering practices. The main message is that life-cycle process modelling creates a basis on which it is possible to distinguish specific viewpoints such as SCM, quality assurance and validation & verification. The ESA/ESTEC process modelling project PMod (Stragapede et al. 1997) provides a formalism derived from SADT/IDEF0 and a solution in which the SCM subprocesses are isolated and described from their own points of view. Examples of these descriptions are given in Figures 16, 17 and 18 (Taramaa 1997).

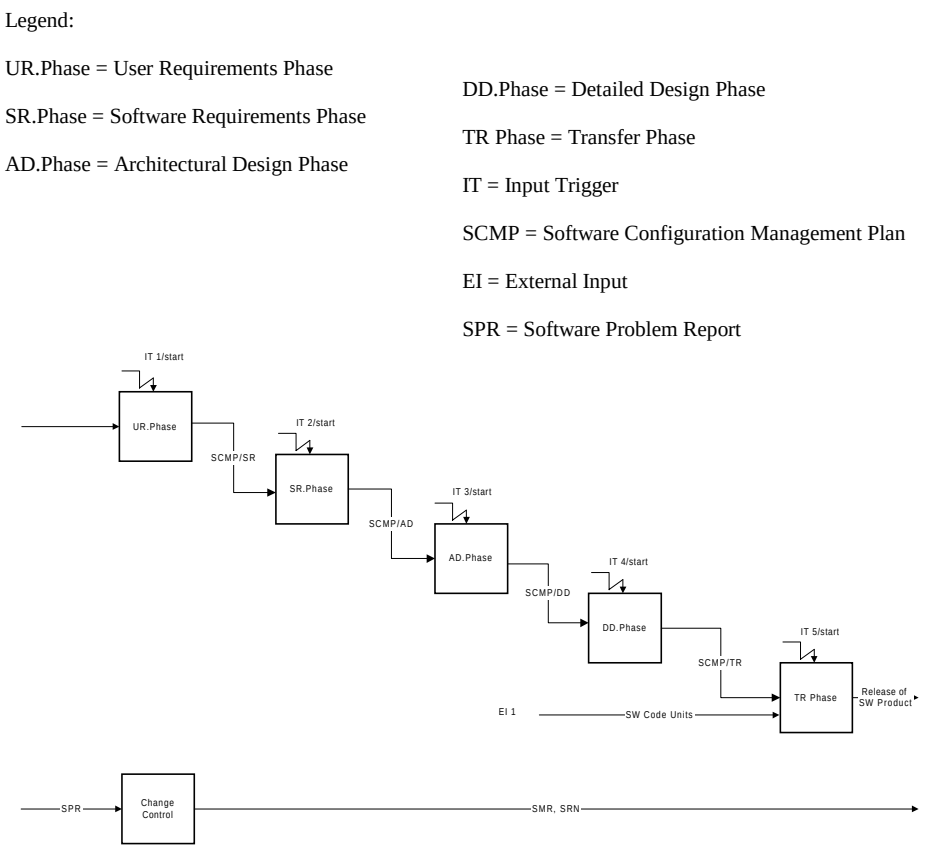

Figure 16. SCM phasewise subprocesses, as recognised in PMod.

Legend: TE = Test Engineer

CM = Configuration Manager HW = Hardware Engineer

SCMP = Software Configuration Management Plan ISVV = Independent Software Verification and Validation

SWPM = Software Product Manager

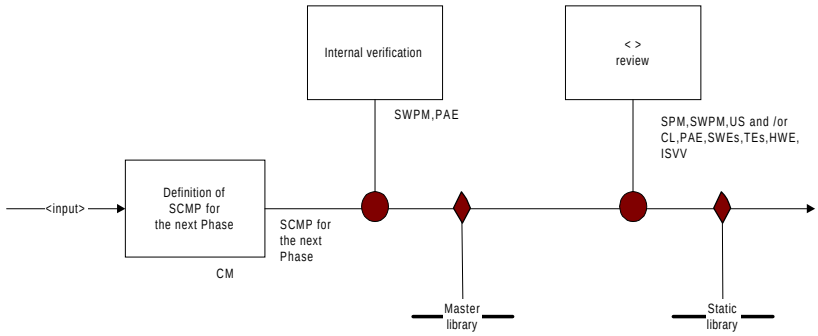

Figure 17. The levelled phasewise subprocesses (e.g. UR.Phase, SR.Phase, .., TR.Phase) of Figure 16.

Legend:

SPR = Software Problem Report *SWE = Software Engineer*

SCR = Software Change Request *SCA = Software Change Analysis*

NCR = Non-Conformance Report *SCO = Software Change Order*

CMO = Configuration Management Office *TE = Test Engineer*

SRB = Software Review Board *ISVV = Independent Software Verification and Validation*

SWPM = Software Product Manager *SMR = Software Modification Report*

TL = Team Leader *SRN = Software Release Note*

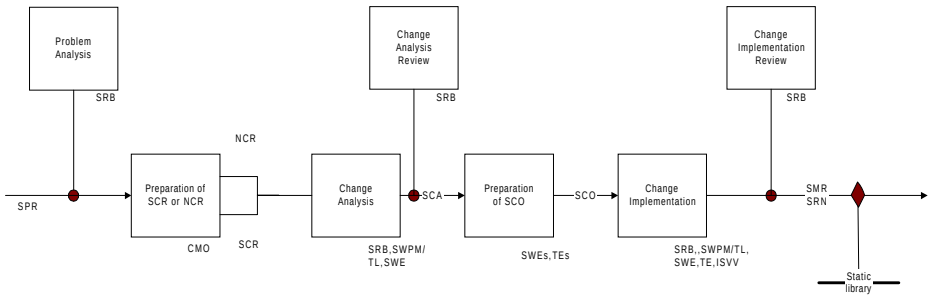

Figure 18. Change control subprocess.

In the ESA's models SCM can be seen in the following ways:

- SCM planning processes (Figures 16 and 17),
- release production, which is dedicated to the quality assurance subprocess,
- change control processes (Figures 16 and 18), and
- version control processes, such as check-in/check-out operations, of which there is an example in Figure 18 described by the symbol connected to "Master/Static library" indicating implicitly check-in operation.

From the point of view this research, ESA's work provides a strong process-oriented approach describing relations between different software processes. Its process descriptions imply a technical solution for version control.

The framework can be seen in the ESA modelling in two ways. On the one hand, the majority of the SCM concepts and elements can be found in the ESA models, and on the other, the ESA models do not include any maturity framework or any improvement paradigm. Like the assessment paradigms, the ESA models do not point to any procedure for implementing SCM.

2.6.3 Software assessment paradigms

Assessment of the software process is becoming a development target of its own in the community engaged in developing embedded systems. Solutions based on quality standards have been developed, many of them conforming to the ISO 9001 standard (ISO 9000-3 1991) and others, especially in the US, being applications of the Capability Maturity Model for Software (CMM) (Humphrey 1988; Paulk et al. 1995), the background to which is formed by the American standard DoD-STD-2167A. The European BOOTSTRAP model (Kuvaja et al. 1994) combines the ISO9001, CMM and ESA models, emphasising process assessment, and conforms to the assessment framework developed by SPICE (ISO/IEC 15504 1995). There is also a domain-specific assessment framework TRILLIUM, developed for telecommunications (Bell 1994).

CMM

CMM, which was produced by SEI for software maturity assessment, is organised into five maturity levels: Initial, Repeatable, Defined, Managed and

Optimising. Each level comprises a set of process goals of capability and is composed of a number of key process areas, each organised into five sections, called common features, which in turn contain what are called the key practices.

The key process areas are defined as residing at a single maturity level, in accordance with Table 6. The basic functions of SCM form one of the key process areas at the repeatable level, as Level 3 consists of set of software engineering functions that include SCM. Therefore, achieving the third level requires documentation, standardisation and integration of SCM-related activities.

Table 6: The CMM key process areas of each maturity level.

Level	Key Process Areas
5 - Optimizing	Defect Prevention Technology Change Management Process Change Management
4 - Managed	Software Quality Management Quantitative Process Management
3 - Defined	Peer Reviews Intergroup Co-ordination Software Product Engineering Integrated Software Management Training Program Organization Process Definition Organization Process Focus
2 - Repeatable	Software Configuration Management Software Quality Assurance Software Subcontract Management Software Project Tracking and Oversight Software Project Planning Requirements Management
1 - Initial	-

SCM is described in more detail in process-specific descriptions of CMM, where its fulfilment is based on the solutions proposed for achieving four goals (Paulk et al. 1995): software configuration management activities are planned and selected software work products are identified, checked and made available. Changes to identified software work products are checked and the groups and

individuals affected by them are informed of the status and content of the software baseline.

Although there are more detailed descriptions available on how to achieve these goals, it is difficult to create an overall view of the relations between the different items. In that sense, CMM and its SCM descriptions do not include any explicit definition of the procedure for finding a solution to the SCM question.

At level 4 the organisation has to set quantitative goals for both software products and processes. Maintainability is one of the product quality characteristics developed at this level. On the process side, process performance information is expected to include SCM. Level 5 is not dedicated to any specific software process, but each separate process activity has now arrived at a stage where it is possible to optimise it.

Compared with the framework put forward in the present research, CMM can be characterised as follows:

- CMM does not define explicitly concepts and elements, such as version control, software manufacturing, change control, since it concentrates on duties which the organisation has to fulfil and which CIs have to manage etc.,
- CMM has a comprehensive maturity framework, where SCM is one of the key process areas, and
- Key process area “Process Change Management” can be regarded as an improvement procedure, although it is independent from any specific process.

SPICE

The other improvement paradigms have borrowed their maturity level ideas from CMM. SCM can be found in all the paradigms. SPICE, which tries to combine experiences obtained in the development of previous improvement paradigms, describes the software processes as comprising development, management, customer support and quality as well as technology transfer. SCM is located among the support processes (SUP.n) in this categorisation, the others being documentation, quality assurance, problem resolution and peer reviews. The subprocesses of CM are:

- Establish configuration management library system,
- Identify configuration items,
- Maintain configuration item descriptions,
- Manage change requests,
- Control changes,
- Build product releases,
- Maintain configuration item histories, and
- Report configuration status.

This means resolving all the SCM levels ranging from version control to change control, although the list does not indicate in which order these items have to be resolved in each environment. In addition, the definition leaves the possibility open to define a detailed solution for each. SPICE can be compared with the present framework in the following terms:

- SPICE defines SCM concepts and elements as work products in more detail than CMM,
- As CMM, SPICE has a comprehensive maturity framework, where SCM is one of the support processes, and
- SPICE has a process area called organisation process (ORG.n), which includes process improvements as one of the subprocesses.

TRILLIUM

A central concept of the TRILLIUM assessment paradigm is a capability area. There are eight capability areas within the TRILLIUM model. Each capability area contains practices, so called roadmaps, which are sets of related practices within the product development process. Each roadmap represents a significant capability for a software development organisation, such as CM. A more detailed discussion of CM activities of the TRILLIUM model is given by Paper VI.

Since TRILLIUM is derived from CMM, it includes many similar features, and thus it is closely related to CM improvement, i.e. the central CM concepts are defined but there is no order laid down for employing neither these concepts nor any technical solution.

2.6.4 Discussion

Since this research examines SCM elements, maturity levels and improvement presenting the solutions creating the SCM framework and tentative SCM maturity improvement levels validating them, these three things have been taken as the columns of Table 7.

The related work has contributed both ideas and practical solutions to this research. The assessment paradigms address SCM. ESA's solutions proposed for space applications provided practical approaches and methods for SCM as a part of the software development and maintenance process. Feiler's and Dart's work to SCM provided ideas for an incremental approach. Feiler's models show the possibility to see SCM by various models, which include parts of the SCM elements. Dart's starting point is that companies have already taken commercial SCM tools into use and their commitment to SCM is therefore very strong. Incremental improvement can be found in the approaches of Feiler and Dart, and also in the assessment-based paradigms, where the maturity levels are based on a more comprehensive maturity context.

A summary of the related activities is provided in Table 7. The related items are derived from Table 4, which illustrates a summary of SCM elements. In addition, process improvement is related to Table 7.

Table 7: SCM features of the related work.

Related approaches	Definition of SCM elements	Definition of SCM levels	Definition of improvement process
Feiler's model (Feiler and Downey 1990; Feiler 1991)	Configuration languages have been left out of scope SCM planning has been left out of scope The strong emphases is in version control, change control, software manufacturing and teamwork	Yes, four levels	No
Dart's model (Dart 1990; 1991, 1992a, 1995)	Configuration languages have been left out of scope The basic SCM elements have been defined by Dart (1991)	Partly related to Feiler's work	Partly, the matters to be taken into account in SCM improvement have been defined
ESA models (Coene 1992; Favaro et al. 1994)	Configuration languages have been left out of scope Version control, configuration audit, change control, software manufacturing have been defined by Favaro et al. (1991)	No	No
CMM (Humphrey 1988; Paulk et al. 1995)	Version control, configuration audit, change control, software manufacturing have been defined in base practices Planning, in general, is the basic concept in the processes to be assessed	No, all are parallel in base practices	Partly
SPICE (ISO 15504 1995)	Like CMM	No, all are parallel in base practices	Partly

3. Requirements from industry

The need to study and develop practical SCM solutions in the field of embedded computer systems has arisen through co-operation with a number of industrial embedded systems manufacturers.

This chapter illustrates the requirements related to SCM of three embedded systems application domains. Paper III shows experiences gained in mechatronics applications, Paper IV concentrates on a space application, and these two applications are related to each other in Paper V. Paper VI presents results obtained with several other electronics products, mainly in the field of SMEs.

3.1 Requirements of machine control software

The first case deals with machine control software. The main focus is to support customer-specific product variations. The central goal to be required is to support the assembly of reusable software components according to the goal of the LOKKI project. The main goal can be further divided into more detailed requirements (Paper III):

- use of modern communication media, sophisticated microcontroller chips and intelligent sensors and actuators has increased the need to manage software more flexibly,
- need to serve customers in increasingly smaller market segments whose special needs have to be constantly maintained, and
- need for repairing and modernising systems many times after delivery.

3.2 Requirements of space software

The second case deals with space instrument software where the main goal was to develop support for application management. The space instrument software development is typically carried out in co-operation with several partners, where ESA or the prime contractor demands a solution for SCM of subcontractor (Murrach 1998).

The cases, in particular space applications of two project partners, created an essential part of the AMES project from which the SCM-related requirements can be derived (Paper IV):

- need for managing several modernisation cycles,
- need to understand relationships with other product technologies in addition to software, and
- need to take account of better application-specific requirements.

These needs have implicit or explicit relations to SCM, which can be made more visible. This thesis relies on the hypothesis that embedded system has to change throughout its lifetime, so that systematic procedures for evolving industrial SCM schemes are also needed.

3.3 Requirements of electronics products

The third case category includes several embedded software applications. The starting point of the LEIVO project was the preliminary version of the incremental improvement framework, which we used in improvement of SCM-related aspects, although the associated companies had different starting level of SCM. In addition, the role of software in this case is more essential compared to the first case because of applications. The requirements of these applications ranged from simple version control solutions to fully automatic PDM systems. The requirements identified at the beginning of the LEIVO project can be summarised as follows (Paper VI):

- need for managing parallel changes,
- need for impact analysis, because the hardware is often designed and implemented simultaneously with the software,
- need for a better understanding of software.

4. SCM framework

The previous chapters outlined a view of the concepts of software maintenance and evolution and of SCM. These concepts include cause-and-effect relations, e.g. the fact that the construction of an assembly system for software manufacturing presupposes solutions for version control.

The framework for SCM practices described in this chapter has been driven by the needs of industry shown in the previous chapter. The incremental development of CM practices ranges from version control to product-level CM, and as each step includes specific features, they are illustrated in detail. The definition of the SCM framework demands a specific procedure, for which the levels are created.

The levels of CM are created on the basis of the results presented in Papers III - V (Figure 19). The first version of the approach was published by Taramaa and Seppänen (1995), whose basic goal was to determine the concepts required for practical SCM and to define their content.

The short description related to Figure 19 is given in this main chapter. A more detailed description is provided by the following sections.

The lowest CM levels in Figure 19 represent *the version control solutions*, with the steps in software manufacturing above them on the left. Their common factor is that they involve features concerning software releases but do not especially deal with evolution based on changes and post-delivery functions. These levels create a basis for the development of software evolution and maintenance practices.

The highest level of *release-oriented SCM* in Figure 19, i.e. automated software manufacturing for mass-customisation, can be regarded as running parallel to change-oriented SCM, since the basis for the development of evolution and maintenance practices has already been created. Software assembly systems in many companies are complicated and their development can be distinguished conceptually from change-oriented SCM, since the solutions concentrate on the management of various configurations, the application of reusability and the construction of advanced user interfaces for finding the right component.

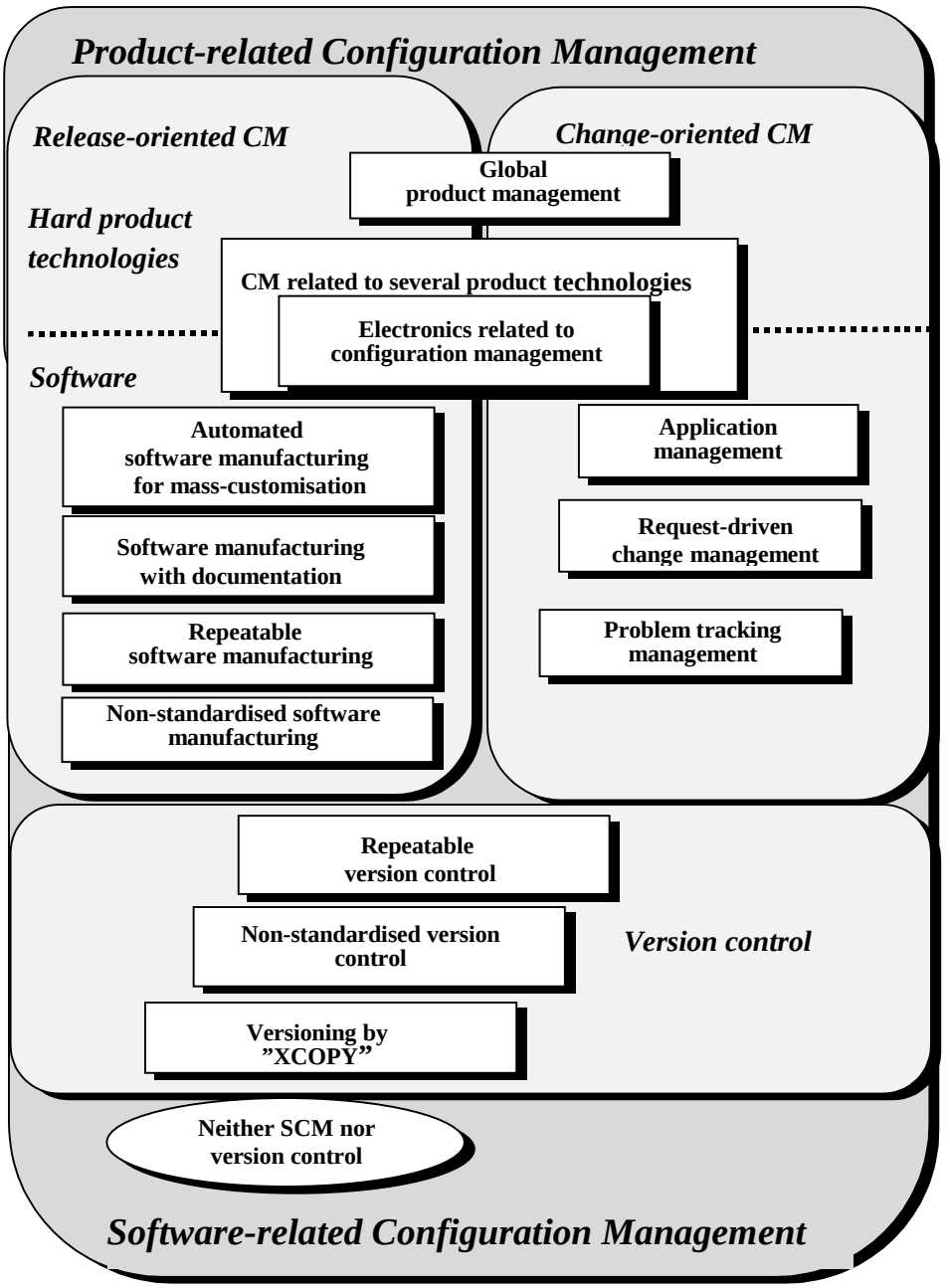

Figure 19. A framework of CM procedures for the development of embedded systems.

Change-oriented SCM should be preceded by at least standardised, repeatable software manufacturing, in addition to which the technical solutions of problem tracking have to be created before any change management practices are implemented. Problem tracking management constitutes the lowest level of change-oriented SCM and lays a foundation for the deep problem analysis required for change management.

Change-oriented SCM can be regarded as parallel with release-oriented software manufacturing, which provides the basic software assembly solutions, since there is no direct dependence between them.

Since software development is typically an iterative process, including the evolution needed for achieving the software to be released, a relation must be demonstrated between release and change-oriented CMs. There is no single, obvious order of the levels, as release and change-oriented SCMs can be partly developed independently and in parallel, depending on the resources available. The development of change-oriented SCM features such as various tools consistently presupposes a certain form of software manufacturing. The next subsections will indicate one of the potential orders that can be used in CM development.

When transferring to the product level the main issue is PDM and the solutions supporting it. PDM systems are typically used to manage hardware-oriented configuration performed by computerised design systems such as CAD/CAE/CAM. The product-level items are important. The integration of software development into PDM in particular requires the construction of interfaces, i.e. CASE descriptions are then seen as existing on the same level as other computerised design presentations. Modern PDM solutions have not yet addressed the question of software-specific parts.

The highest level shows solutions for distributed configuration mechanisms, which become necessary when the providers of computerised design results are to be located geographically in different places.

4.1 Version control

Copying, e.g. a PC/DOS command XCOPY, represents the lowest-level manual solution for SCM, and the quality of the results depends greatly on the experience and personal capabilities of the software engineers.

Version control provides a means to maintain specific instances of software products developed for different environments, and its use is supported by the saving achieved in memory space and formalised identification practices. Based on experiences gained in two joint projects with Finnish industries, the common practice seems to be that the version control tool is used differently in separate software development projects for the same company. This can be called non-standardised version control. This will, in practice, introduce new problems when parts of the software are needed in other projects, or when the CM process is being improved. Practical experiences with work on embedded systems version control are recounted by Sperry (1992) and Kooshian (1994). The development of a companywide quality system can be supported by making guides for the use of version control, i.e. ”standardised and repeatable version control”. This can be regarded as a more advanced level (Paper IV).

The embedded features of embedded software can be seen as specific CIs which have to be controlled, such as software-hardware interface documents, hardware-related modules, simulation results as a part of prototyping, images as a result of software manufacturing.

Related to version control, safety and security concerns are reflected in configuration audit, which sets strict requirements to verification and validation mechanisms. These mechanisms provide input for decision making of configuration audit often demanding the fulfilment of various safety and security standards.

4.2 Release-oriented SCM

Release-oriented SCM concentrates in software manufacturing where it is possible to find solutions at different levels ranging from simple individual,

non-standardised until automated, comprehensive software manufacturing mechanisms.

4.2.1 Non-standardised and repeatable software manufacturing

The activities for producing a software release create the own levels of SCM framework. These levels deal with improvement of software manufacturing, i.e. to integrate version control into a software assembly system that contains configuration builders such as compilation and linking tools. Automatic assembly requires the management of a variety of different revisions of files by means of a configuration builder. The most common method of dealing with multiple versions is to add labels to a set of file versions and to compile the whole set using a batch command. Solutions based on makefiles are common in UNIX environments. The programming language-oriented environment such as Borland C++ environment already provides a build mechanism. Examples of the use of makefiles are given in Paper III and by Flynn (1995).

From the viewpoint of embedded software, the main benefit of using automatic configuration builders is to control the complex compilation process. In some cases the software must be pre-processed through three or even four compilers, using different parameters.

Nowadays it is easy to start using SCM systems with build mechanisms, because several such systems are commercially available (Forte 1993). Problems appear if there are several different SCM systems in the same company. As in the standardisation of version control as repeatable, CM also benefits from standardised practises.

4.2.2 Software manufacturing with documentation

The management of various life-cycle documents to be produced during software development is a natural part of software assembly. Typical documents are specifications, designs, user guides and test specifications and results. Embedded software-specific documents are software-hardware interface documents and simulation specifications and corresponding test results.

In general, all baselines produced during the life-cycle create a software item in configuration. To separate the production of different life-cycle documents from these items would be very laborious, as the documents would not follow the same baselines as the software. The various documents are very often seen as an intermediate result in software development and maintenance. The creation of this practice requires an effort of its own to keep the documentation of all the life-cycle aspects at the same level as the software. The incorporation of documents to configuration descriptions is therefore a consequence of progress in the document management process.

4.2.3 Automated software manufacturing for mass-customisation

Reusability and distribution are important questions in software engineering, and they can be related to SCM once the basic SCM problems such as version control and manufacturing have been solved. They are independent development targets and can be regarded as parallel from the point of view of SCM.

Commercial SCM systems nowadays provide solutions for manufacturing binary files, but companies need specific, tailored solutions for their own environments. This is typically the case with high-volume products in which there are small differences from one order to another and manufacturing is based on a combination of reusable software components and components with new features developed especially for a new specific need.

New software life-cycle paradigms such as prototyping also place similar demands on SCM as reusability, since the result of prototyping may be a component, which has to be linked to the rest of the software for early simulation.

If the embedded software packages to be delivered as parts of products differ from each other, there will be a need to develop practices for collecting the right features for each order but in such a way that the details of the version control mechanism remain hidden. Typical solutions are based on commercial SCM systems, which create the core around which companies construct product-specific features. Knowledge-based solutions have also been shown to be useful in these situations (Hakkarainen et al. 1988). They are also recommended, e.g. Cagan (1994).

Like document management in software manufacturing, reusability forms a specific dimension on which the ways of producing, finding and using reusable components vary considerably. Reusability ranges from design patterns (Buschman et al. 1996) to code generation and parametrization.

In addition to reusability, distribution aspects impose requirements of their own at this level. In distributed applications there may be a need for dynamic configuration of separate nodes, and various MIL languages have been developed for this purpose, languages that contain support features for distributed applications in addition to SCM activities, e.g. support for multilingual and dynamic re-configuration.

4.3 Change-oriented SCM

Change-oriented SCM concentrates in change control or change management where it is also possible to find solutions at different levels ranging from simple problem until application management where the maintenance process is implemented comprehensively.

Change management is a specific task that is often conceptually separated from SCM activities and referred to as the problem/software resolution process. According to definitions (ISO 12207), the problem resolution process is a process for analysing and resolving the problems.

In the most advanced SCM environments change management is included as an inseparable part of the SCM scheme and implemented by means of problem tracking systems which has been also applied in this framework.

4.3.1 Problem tracking management

Problem tracking is the first level of change-oriented SCM. Its solutions range from a simple problem, the recording of an error/bug or defect, to advanced databases, which are, related to the other SCM activities. In simple solutions the recorded error data are used manually in software change activities, whereas the advanced database solutions are an essential and integrated part of the other aspects of SCM. The lowest-level problem tracking systems run parallel to

release-oriented SCM. In addition, the problem tracking system is a predecessor to request-driven change management.

Modern commercial SCM systems are being expanded by incorporating problem tracking as one feature in them (Forte 1993). Another feature is the connection with process improvement mechanisms, since the problem tracking database is a central element as a source of metrics information when developing process improvement practices.

4.3.2 Request-driven change management

Up to this level version control and SCM incorporate different features. The next level involves the creation of a mechanism for maintenance, i.e. for integrating version control and software manufacturing with change control. We call this request-driven change management. A typical change management system is an in-house database in which change requests are recorded and managed. Often the data base structure is not well organised and does not support any problem/defect tracking.

The long development and maintenance phases of embedded systems imply that the persons maintaining the system are most likely not the ones who developed it. In addition, the initial system documentation may no longer match the actual design and code, which further complicates the maintainer's problem. Different pre-maintenance tools, such as problem understanding, reverse engineering, traceability and impact analysis tools, can also be used. Their use is currently based on the maintainer's own ability to apply them.

4.3.3 Application management

Application management is the extension of SCM by taking the maintenance process into account more comprehensively. The efficient implementation of application management requires process-based solutions, e.g. the examples given by Taramaa and Ketola (1995) based on the work done in the AMES project (AMES 1995).

At the application management level, specific tools are used for subfunctions of the maintenance process such as problem understanding, localisation, impact

analysis and solution analysis which already acted separately in request-driven change management. The main extension compared with request-driven change management is related to process aspects and management at a higher level of abstraction. The process-oriented approach provides an opportunity to manage the use of tools more efficiently than in request-driven management, where the user still has to activate separate tools.

The process approach is a hot topic today. Most process-driven approaches are focused on modelling sequential events, and are typically prescriptive. This kind of process may appear to be restrictive in nature, however. In the process thinking it can be called control-oriented. The process should be more descriptive, providing guidelines for different maintenance situations. Such a process can be called data-oriented.

4.4 Product data management

Product data management can be examined at two levels: management of software and hardware, and management of the whole product, including all product and manufacturing/production technologies.

Typical features facing development teams in product management are:

- location of different product data,
- integration of different PDM and related systems,
- data transfer, its logical and technical alternatives,
- various implementation alternatives.

Figure 20 illustrates the alternatives of the relations between PDM and SCM systems. This figure depicts different practical aspects of these environments, i.e. location of product data and distribution of PDM and SCM systems. In the environments the following alternatives can be found:

- There are various implementations of PDM and SCM locating in different sites and there is need for data transfer (connection 1 between PDM and SCM, and connections 2, 3, 7, 8 and 9 to various implementation).

- There are separate implementations of PDM and SCM locating in different sites and there is need for data transfer (connection 5 between PDM and SCM, and connections 4 and 6 for distribution are implemented by environment-specific transfer mechanisms).
- There is a common implementation of PDM and SCM locating in different sites, where data transfer has been implemented by a common PDM system (connection 10).

4.4.1 Electronics related to configuration management

In the telecommunications field, for example, computer-aided implementation can produce embedded software for digital signal processors and hardware implemented by FPGAs and ASICs for use alongside software implemented by the traditional processor technology. The differences can be found both in CIs, which include a large number of simulation descriptions in addition to normal algorithm models, and in manufacturing, which is based on VHDL. When transferring to system-on-silicon principles, control software is one of the technologies to be used to create the final product.

According to Soininen (1995), less attention has been paid to research into change control in the context of hardware descriptions. The change management tools applicable to maintenance are different because of the descriptions to be updated. In addition, the different simulation descriptions, which are essential in hardware design, entail specific features in the construction of change management.

The concurrent engineering solutions adopted in co-design represent an extensive challenge to process modelling. Heikkinen (1997) addresses the issues that need to be taken into account when developing a SCM environment which supports these complex process features, i.e. prototyping and assumption reconciliations. Although these items cover software and hardware functions, the mechanisms can also be applied when transferring to deal with concurrent engineering related to other items.

The situation between electronics design and SCM is illustrated by connection 1 of Figure 20. Electronics CAD systems create one of the PDM related CAD/CAM/CAE systems.

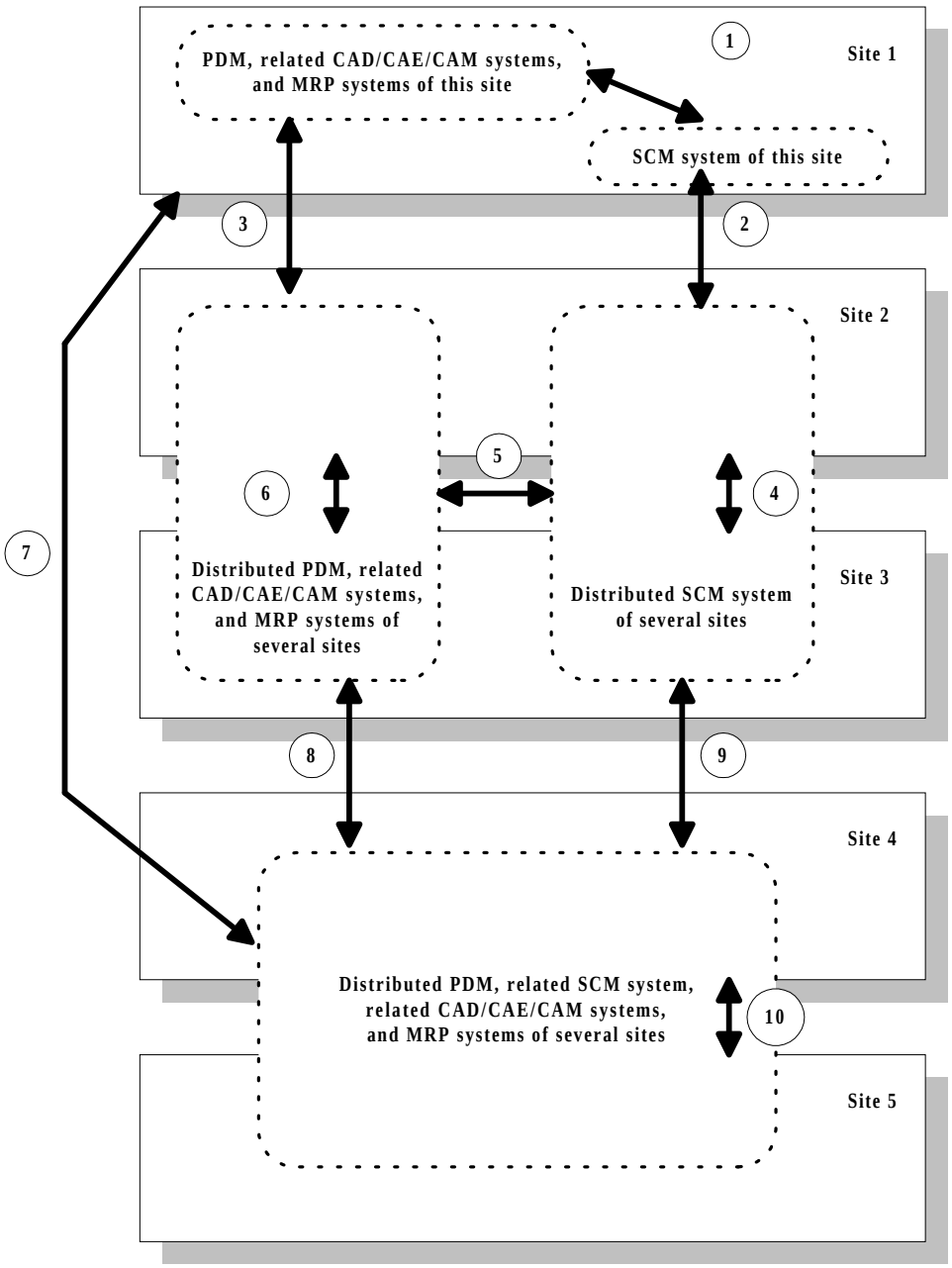

Figure 20. PDM and SCM solutions with various implementation alternatives.

4.4.2 Configuration management related to several product technologies

Considered in a wider framework, embedded systems also include other product technologies, such as electrical and mechanical parts designed using specific computer-aided methods and tools (Paper II). The integration of a number of computer-aided tools, which support other development technologies and logistics functions, places SCM developers in a totally new situation.

These technologies can also be developed under the heading of concurrent engineering. One of the first European projects devoted to product-level concurrent engineering is CONSENS, which can be regarded as promising because of its starting point, that there are already viable and usable design technology-based tools which need better formalism for information exchange (Singh and Irvine 1995).

In Figure 20, site 1 illustrates the connection of one site between PDM, its related and SCM systems.

4.5 Global product management

Distributed configuration mechanisms become necessary at this level, since providers of computerised designs may be located geographically in different places.

The concept of virtual corporation has been presented as a solution to the question of global product management (Davidow and Malone 1992). The virtual corporation has been also recognised as essential in software engineering society (Boldyreff et al. 1996; van der Hoek et al. 1996; Noll and Scacchi 1997).

Figure 20 illustrates different possible relations of the heterogeneous PDM and SCM environments. Site 1 of an environment indicates separate technical solutions of PDM and related computerised implementations. Site 2 and 3 show common solutions by which the PDM and related systems have been implemented in several sites.

In Figure 20, from the SCM point of view the essential relations can be found between SCM and others systems of site 1 (connection 1), between different SCM systems of separate sites (connection 2), within the distributed SCM environment (connection 4), between distributed as well PDM as related systems to the distributed SCM system (connection 5), and PDM and SCM within the PDM system (connection 10).

4.6 Discussion

The elements of the lowest SCM levels can be implemented by buying SCM tools, including version control and building mechanisms, and by defining the usage principles in a companywide software quality system. In addition, problem tracking systems have been commercialised and are required to link their solutions to the rest of the SCM. These solutions create a basis for the more advanced SCM practices. The main effort can then be invested in change-oriented CM, automated mass-customisation and further product-level integration.

Automated SCM based on mass-customisation is assuming a still more central role, in which the main problems to be solved are the role of design patterns, code generation and modelling of various features. All of these are background issues lying behind a sophisticated SCM system.

Change-oriented SCM consists of problem tracking, request-driven change and application management. The main difference when transferring from request-driven change to application management is the role of the software process. Application management is strongly characterised by the process viewpoint, which is adopted in order to manage various change situations.

Although the SCM framework has been presented in terms of levels, these levels have to be tailored separately for each according to the nature of business. Mass- customisation, for example, can assume slightly different role in each company. One company may produce a lot of similar products with very small changes in their functions, whereas another may develop products, which are highly customer-specific, with a small number of common functions. Many companies have made decisions in the past, which restrict their room for

manoeuvre in the future. This typically concerns technical solutions, which restrict the available alternatives for future SCM environments.

5. Improvement of SCM

The importance of SCM is understood in industry, but there is as yet no clear guideline on how to proceed when improving SCM practices. One of the biggest problems is that most organisations have considerable existing software assets to be maintained and this software has almost invariably been produced using a large number of different methods, languages and tools, which makes its management difficult (Paper VI).

Our approach is characterised by co-operation with companies which do not themselves provide fundamental tool or method support for software development and maintenance. Many companies are either SMEs or small software engineering groups within larger companies. Companies with limited software resources have started to employ SCM as a part of general quality engineering practices only within the last few years. The starting point may be very low, at the XCOPY level, as indicated by the project lying behind Paper VI, for example.

On the other hand, there are a number of companies with software engineering staff ranging from tens to hundreds of people (Paper I). These have typically noticed the importance of industrialised software development and maintenance by the mid-1980s improving separately SCM elements of their own software engineering environments (Kemppainen 1986; Mukari 1988). SCM has been one of the targets of this development (Hakkarainen et al. 1988; Lehtinen 1994; Viskari 1995), which explains the higher starting level of some companies in the improvement of SCM.

The SCM framework presented in the previous chapter can be regarded as a descriptive model supporting the understanding and analysis of current SCM practices. However, the improvement procedure to be shown below will establish a starting point for the detailed improvement of SCM practices.

Since improvement can be directed at several SCM and related elements, these elements have been first related to each other. The other part of this chapter is focused on an improvement procedure PR^{2}IMER that provides a general approach for various processes to be improved.

5.1 Overview of SCM improvement levels

The improvement envisaged here consists of maturity levels defined by SCM-related elements. The improvement of the SCM-specific elements is partly parallel. Therefore the SCM elements have to be presented by the parallel columns so as to show both parallel and incremental improvement. In addition, we have included logistics-specific features as characterising SCM-related elements to maturity improvement, because they have to be resolved in any comprehensive CM solution. This was already indicated by the experiences presented in Paper III.

In practice, the incremental improvement of SCM starts with an assessment of the SCM process to determine current practices (Paper IV and chapter 4). This is shown in Table 8, which creates a basis for the assessment of the LEIVO project covering several companies at different levels (Paper VI). In addition to version control, software manufacturing and change control, the assessed aspects include order/delivery and customer data management.

These tables are based on previous mentioned experiences with the industrial partners. The first version of the approach was published in Paper VI. The following tables are more formalised and better related to the current presentation than the first version given in Paper VI. The step numbers were defined in the early phase of the LEIVO. Paper VI and this thesis use this division of the steps, although the steps of Total product management have been combined.

5.1.1 Version control-oriented levels - from step 1 to step 3

The levels presented in Table 8 are named according to SCM elements, although they include the development of logistics functions in parallel with CM. The lowest level is called "No product management", which means a largely manual process without any formal storage of configurations, errors or logistics data. When transferring to "Copying activity with manual versioning" version control is assumed to be based on a computerised system and information on the other related elements are recorded manually. At the next level, "Basic version control", the companies are using various version control systems without standardised or repeatable features. In software manufacturing the computerised

assembly systems are based on batch files and the logistics will show the first computerised solutions. On the "Repeatable version control with makefile mechanism" level, version control is standardised and the components in the archive contribute to one project. The solutions in software manufacturing are based on makefile builders, which are under version control, and the problem database at this level is also linked to source files, which are under version control. In the logistics-specific functions the order/delivery and customer data are recorded in specific databases.

5.1.2 Software manufacturing-oriented levels - from step 4 to step 5

At the next SCM maturity levels the development is concentrated on software manufacturing and change control. At the starting level, "Basic configuration management", software manufacturing is totally under version control. The technical solutions adopted in software manufacturing can still differ depending on the product.

Table 8: The lowest levels of the SCM improvement and CM-related elements.

	SCM-related elements				
	SCM-specific elements			**Logistics-specific processes**	
Levels of improvement	**Version control**	**Software manufacturing**	**Change control**	**Order/Delivery data management**	**Customer data management**
No product management **0**	No archiving Backup system may allow return to older versions One "latest" version	Manual building of products One "latest" version	No problem/error lists	No information stored on deliveries Invoice & bookkeeping the only source of order/delivery information	Invoice and bookkeeping the only source of information
Step 1					
Copying activity with manual versioning **1**	Manual archiving using packaging (ZIP) or copying (XCOPY) Source codes for the main delivery versions are stored in a common project directory	Lists of configurations	Manual problem/error lists	Manual order/delivery file on paper	Manual customer files on paper

Step 2					
	SCM-related elements				
	SCM-specific technical elements			**Logistics-specific processes**	
Levels of improvement	**Version control**	**Software manufacturing**	**Change control**	**Order/Delivery data management**	**Customer data management**
Basic version control **2**	Product has its own archive All source revisions retained and accessible Design documentation and user documentation under version control	Change information on each CI is documented manually Batch files used in builds Release is produced automatically from version controlled code All releases and test versions are produced automatically from version-controlled code	Computerised problem/error data base Problem/error counting metric	Computerised order/delivery register Order handling & invoicing systems (purchase order systems)	Computerised customer register
Step 3					
Repeatable version control with makefile mechanism **3**	Archive organised into components Components designed to be reused within one project	Products are built using makefile mechanism Makefile is under version control	Problem/error base with problems linked to source file Problems are reported and collected directly to specific product and module versions.	Order/Delivery database	Database contains delivery information All customer contacts are recorded
Step 4					
Basic configuration management **4**		Makefile dependencies reach all the way to version control		Order/Delivery data contain detailed information on products supplied	Service request concept formalised

Step 5					
	SCM-related elements				
	SCM-specific technical elements			**Logistics-specific processes**	
Levels of improvement	**Version control**	**Software manufacturing**	**Change control**	**Order/Delivery data management**	**Customer data management**
Repeatable configuration management **5**	Project management documentation and quality documentation under version control	Automated dependency generation	Problems linked to specific revisions of source files Problem classification system used to produce feedback to inspection, testing & coding	All orders/deliveries to each customer can be traced , to individual source code modules	
Step 6					
Basic product management **6**	All versions of build tools retained All documentation including hardware and other product technologies in a version-controlled database	Build environment configurations are described in makefiles Build configuration is stored in database Build record database (stores source file revisions used in each assembly)	Problems linked to specific revisions of all affected documentation, such as source files, change requests, specifications, etc. Problem reports are used to optimise the residual problem level and testing Request-driven change control	Delivered products can be fully rebuilt later	

Steps 7-11

	SCM-related elements				
	SCM-specific technical elements			Logistics-specific processes	
Levels of improvement	**Version control**	**Software manufacturing**	**Change control**	**Order/Delivery data management**	**Customer data management**
Total product management **7-11**	Software components are reusable across several products The whole product is configured automatically	Automated configurable assembly systems Build configuration tools of all assemblies are uniquely identifiable Assembly systems are based on MIL languages supporting distribution	Problem reports are used to optimise the software process Application management-oriented change control	Delivery database with links to build records and build configuration records Information about customers' environments is stored	Service request contains links to all affected source, design and product management files
Step 12					
Global product management **12**	Components are reusable across several companies Software components are only one category of product components	Automatic configuration of third party products Software configuration is only one assembly category			

In "Repeatable configuration management", version control can be extended to cover project management and quality documentation. The automation used in software manufacturing is comprehensive, i.e. all the assembly activities are predefined using the make file mechanism and version control. The treatment of error data is more detailed at this level, i.e. problems are linked to specific revisions of the source file. The solutions to logistics functions at this level are linked to the source code supplied.

5.1.3 Change-oriented level - step 6

At the "Basic product management" level some items such as build tools and part of the documentation can still lie outside version control. A comprehensive version control is required in order to achieve this level. The extensions of software manufacturing deal with the use of a more flexible build mechanism and a database that includes all the configurations. Change control includes the first solutions for supporting software evolution and maintenance, although the tools are separate entities. The problem tracking management solutions are linked to software documentation, and the problem tracking system is beginning to be used in the optimisation.

5.1.4 Total product management level - step 7 to step 11

The total product management level covers efficient usage of reusable components, an automated, configurable assembly system and applications management.

The first two features form the basis for automated software manufacturing, including mass customisation. Applications management includes extensions of change control in which the tools are related to a predefined process description. This level is not subdivided in Table 8, but one possible division is presented in Figure 1 of Paper VI. At this level the solutions to the logistics functions are closely related to the SCM solutions.

5.1.5 Global product management level - step 12

The geographical distribution is one of the key factors when transferring to global product management. The main impacts of global product management were indicated to be PDM, related CAD/CAE/CAM, and MRP systems, which are in the focus of Table 8.

5.2 Improvement procedure for SCM

The improvement to SCM applied in this thesis is based on an inductive approach in which the starting point is an understanding of the organisation and its problems in order to set goals for making improvements. The understanding is based on the SCM framework by which separate SCM elements can be understand better in an industrial organisation. The PR^{2}IMER (Practical Process Improvement for Embedded Real-Time Software) approach for developing application management solutions (Karjalainen et al. 1996) as presented in Paper IV (Figure 21) can be regarded as an instance of the PR^{2}IMER concept, since it did not employ all the PR^{2}IMER features.

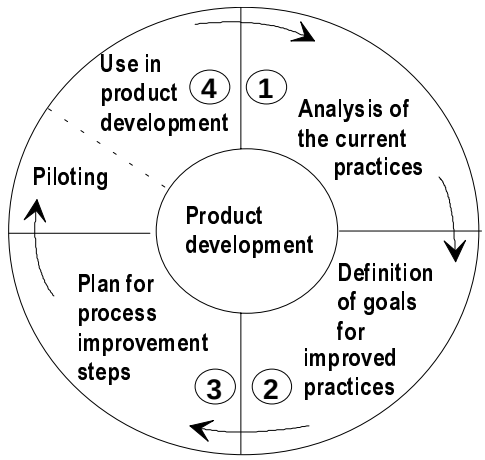

Figure 21. Process improvement using the PR^{2}IMER framework (Karjalainen et al. 1996).

PR^{2}IMER integrates software process analysis, modelling, improvement, and measurement techniques to meet the critical quality requirements for developing embedded systems. It consists of the following steps:

1. Quantitative and/or qualitative analysis of current software development practices,
2. Definition of measurable goals for improving those practices,
3. Planning of successive and practical process improvement steps,

4. Piloting and trial use of the improved practices in product development projects.

The purpose of software process analysis is to describe current software development practices, to identify problems and to define objectives for improvements. In our case, the objective was predefined as improvement of the SCM/CM process. In addition, current SCM/CM practices were analysed using the SCM framework presented in this thesis. The problem was therefore focused on elements defined by the SCM framework.

The problems identified and requirements stated during the analysis of current software development practices create the basis for the definition of measurable goals. The GQM-based approach provides a way to develop new practices in order to achieve the goals set and to produce metrics for measuring the improvement in the process.

Since software process improvement measures are implemented stepwise, it is necessary to have a clear view of the process to be improved. In this case the SCM framework provides a starting point for analysing and understanding the separate SCM elements. The improvement itself consists of SCM-oriented elements, including specific technical solutions in accordance with the previous section.

The revised software development process and related methods and tools were further adopted in pilot projects. Improvements were measured according to the GQM-based measurement.

6. Validation

This chapter illustrates validation of the SCM framework and the first improvement experiences.

The SCM framework has been applied in three embedded systems application domains. Paper III describes experiences gained in mechatronics applications, Paper IV concentrates on a space application, and these two applications are related to each other in Paper V. Paper VI shows experiences of electronics applications.

In addition, the first improvement experiences have been provided via two applications, space and electronics. Paper VII shows quantitative measurements of tools applied in space applications. Paper VI presents results obtained with several electronics products, mainly in the field of SMEs, related to maturity improvement levels.

6.1 Cases in SCM framework validation

The SCM framework validation was based on three cases:

- SCM solutions in mechatronics applications (Papers III and V),
- Application management for a space application (Papers IV and V), and
- The more comprehensive validation with other applications of electronics products (Paper VI).

In parallel, the SCM framework has also been applied in the EU/ESSI (European Software System Initiative) project TPM (Total Product Management in Technopolis) (Kilpi 1996).

6.1.1 Mechatronic application

The starting point was the lowest SCM level, XCOPY, and improvement was based on the use of a commercial SCM system PVCS. Although version control was used by the software itself and its relation to the quality system was understood, insufficient guidance had been produced for its use. The main effort was directed at the development of software manufacturing features for release-oriented SCM. The prototype environment provided support for the configuration of assembly systems. The work also produced the first results of reusable software for the further improvement of release-oriented SCM.

The other parts of the development framework were beyond the scope in this case, but order/delivery management and customer data management were observed to be important elements of SCM development, e.g. the links between product-level and SCM were recognised. The case disclosed the need to develop the PDM system related to SCM. This result created a starting point to the ideas, which enabled further development of the SCM framework.

The development of basic solutions for version control and advanced solutions for software manufacturing in relation to the requirements provided an opportunity to manage the software better. Table 5 is a result of this development illustrating the potential assembly alternatives, i.e. configurations.

As a part of SCM development, a demonstration environment was constructed for a mechatronics company (Paper III), in order to extend the separate commercial tools to take into account requirements dictated by the application, such as use of version control, documents and software to be archived, construction of configurations and role of a build mechanism, as shown in Figure 22.

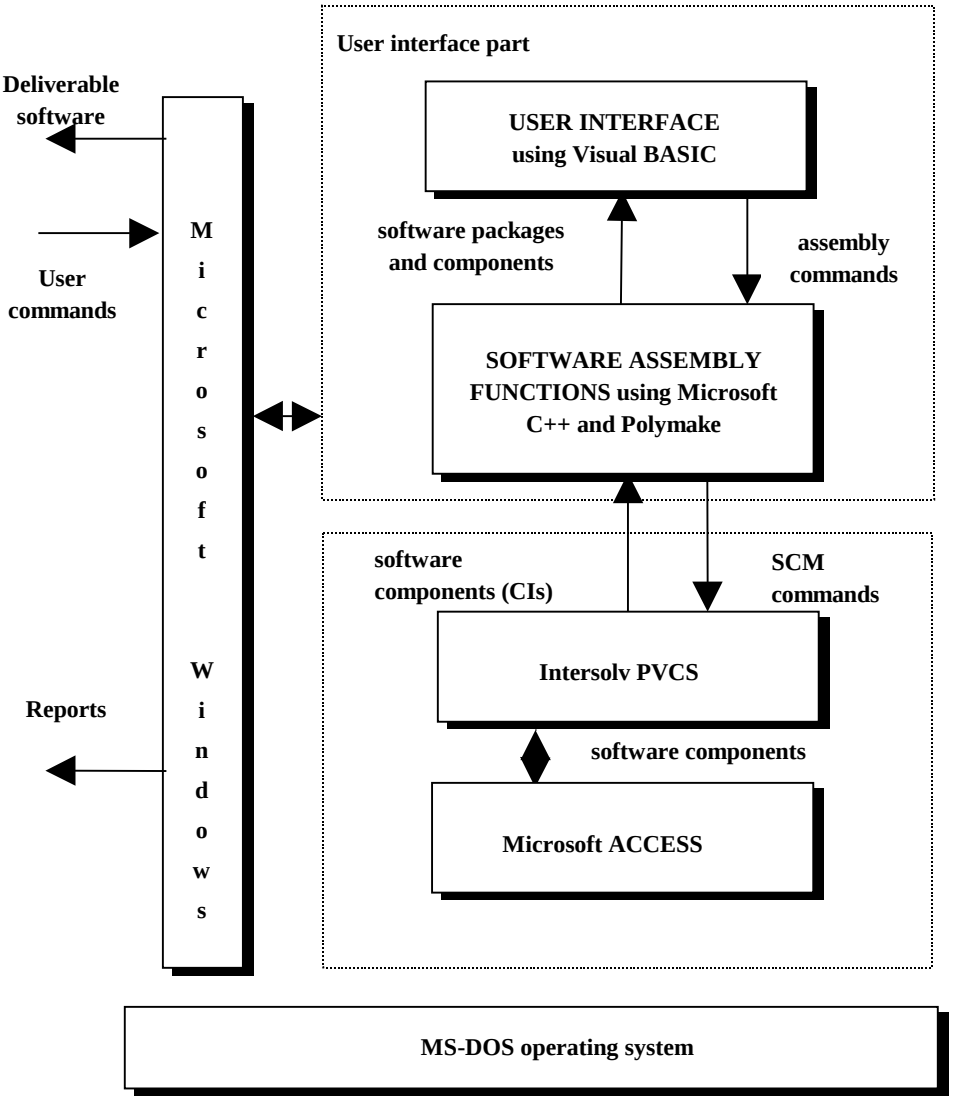

Figure 22. A prototype of an assembly system.

6.1.2 Space application

Space applications are evolutive, since development takes many years and includes numerous adaptive, corrective and perfective changes before delivery, after which changes are difficult to implement.

The space applications form a specific area in software manufacturing, since the embedded software to be developed is unique. Normally it will be not reused in company after delivery. The reusable aspects are not so important because of the unique nature of the developed software. The major emphasis is in change control during the evolutive software development.

This application displayed a repeatable version control level, since both software and design and user documentation were under version control. Software manufacturing in the context of release-oriented SCM was not an important, since a simple assembly system based on SCCS and makefile was sufficient for this application area. Order/delivery management and customer data management were beyond the scope of this case.

The application management architecture was developed as a part of the AMES project in accordance with Figure 23. The work concentrated on defining and implementing the application management process and the tools to be developed.

Because the AMES project belonged to the ESPRIT programme, it included the emphases favoured by all the partners. The author and the colleagues of his home and Finnish case company concentrated on a space application and the construction of the AMES environment according to the requirements of this application. The case study provided input for defining the AMES environment and the relation between the SCM and maintenance concepts. Papers IV and V give a more detailed description of the development work. Finally, the applicability of the resulting tools was evaluated, as presented by Laitinen et al. (1997) and Paper VII.

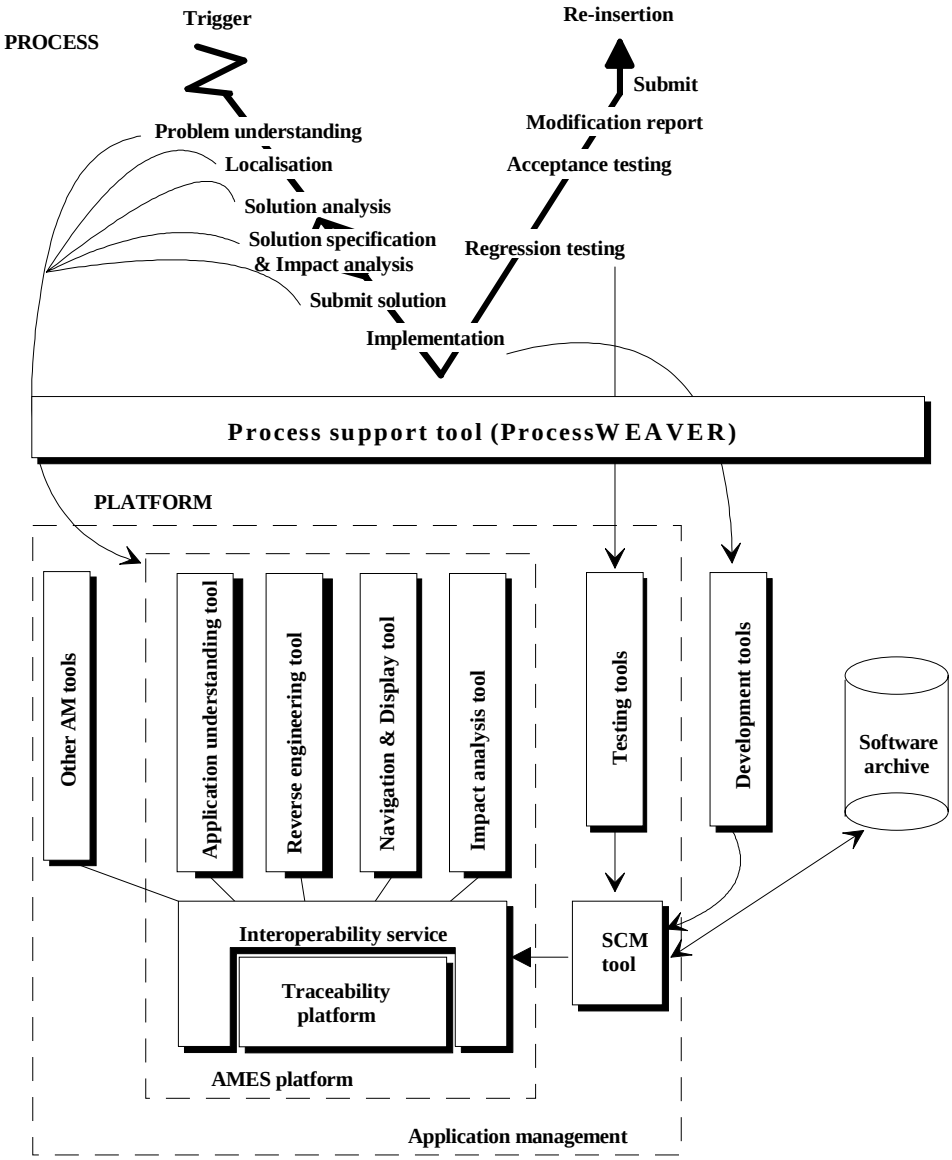

Figure 23. Application management architecture.

6.1.3 Other applications of electronics products

The final case studies were made in several companies with applications in electronics, telecommunications and machine and process automation. These case studies provided the input for the development of the revised maturity framework to be presented in the next chapter.

The starting point here was to analyse the current status of the SCM practices according to the PR^{2}IMER framework. This provided an overall view of the SCM elements. Since several companies with quite different SCM requirements participated in the project, we first had to establish a general view of current SCM practices. With this in mind, the SCM framework provided a way of analysing SCM practices. More detailed requirements for SCM improvement were formulated during the project.

6.1.4 Discussion of SCM framework validation

The above applications and their SCM development showed that it is necessary to understand each main SCM element and to deal with these elements separately. In most applications the separate improvement of SCM elements was unavoidable, since:

- Insufficient resources were available for parallel improvement of the SCM elements.
- There were dependency relations between SCM elements. E.g. version control and software manufacturing have to achieve a standardised procedure before an efficient improvement can be achieved in change control.
- Improvement of software manufacturing also requires advanced solutions with regard to documentation. Linking documents to configuration requires procedures for guaranteeing the consistency of documents with software.
- Improvement of change control requires a documented maintenance process in each company.
- Transfer to product data management requires that SCM solutions should have been reached with regard to version control, software manufacturing and change control.

6.2 Initial improvement experiences

The SCM improvements were validated in two projects, AMES (AMES 1995) and LEIVO (Paper VI). The first only concentrated on change control as a part of application tools and management for space applications (Paper VII). The other concentrated on version control, software manufacturing and order/delivery data management in connection with a number of electronics products (Paper VI). The validation of the first project was based on use of qualitative analysis of current practices, GQM for goals definition, quantitative measurements derived from GQM plan. The validation in the other project was based on qualitative evaluation.

6.2.1 The AMES project: Improvement of change control

The functional relationships perceived in the development of the change control-oriented environment are shown by Figure 24. The AMES project proceeded in three lines: applications analysis, tools development, and application management process development.

Applications requirements derived from application analysis created a basis for tools evaluation, which were made using scenarios, which described briefly the evaluation steps. The process development was one of the tasks. It was based on the previous ESA's process modelling work. The main effort of the AMES project was invested to tools development (section 2.4.1) where one of the tools should be a connecting part for all AMES and other tools covering process aspects. The process support was not successfully implemented and it could have not been really tested.

Analysis of the current practices

The preliminary version of the PR^{2}IMER framework was used to develop a space application, as a part of the AMES project. The analysis of current practices was based on a qualitative method (Mäkäräinen and Taramaa 1995), involving focused interviews with developers of both product and software development environments. A sample of these results is shown by Paper IV.

Definition of goals for improved practices

The definition of improvement goals was based on requirements formulated together with product developers (Mäkäräinen 1996). The plan for improving applications management was implemented by scenarios where the application understanding toolset was one of the evaluated tools (AMES 1994b).

Improvement and its analysis

The process enactment tool ProcessWEAVER (Fernström 1993) was used in the pilot phase in accordance with Paper IV. The comprehensive view of the prototype environment illustrated by Figure 23 provided support for application management implemented with several specific tools. At the very end of the project, the benefit was assessed via these tools using the GQM method (Paper VII). According to GQM, defining metrics starts from goal definition as indicated in Table 9.

Table 9: *The basic template for the goal definition in GQM.*

Basic statement:	Instantiation:
Analyse	the products (AU, ReverseNICE, IAS)
In order to	evaluate them
With respect to	the tool value
From the viewpoint of	the maintainer.

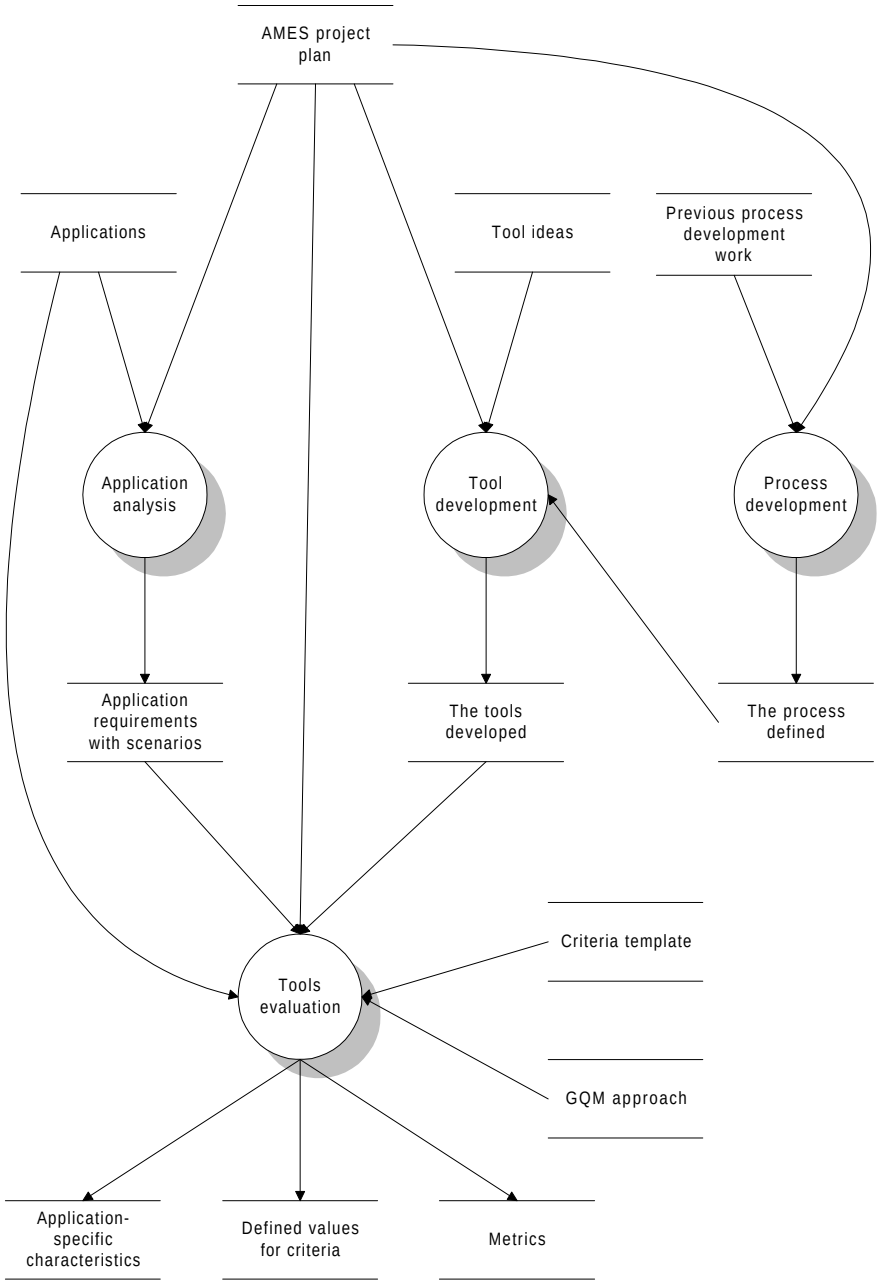

Figure 24. Tools evaluation as a part of the change control improvement project in AMES.

The evaluation was based on a framework, which included:

- criteria, and
- goals with explanatory questions and specific metrics.

The structure of the relationship is described in Figure 25.

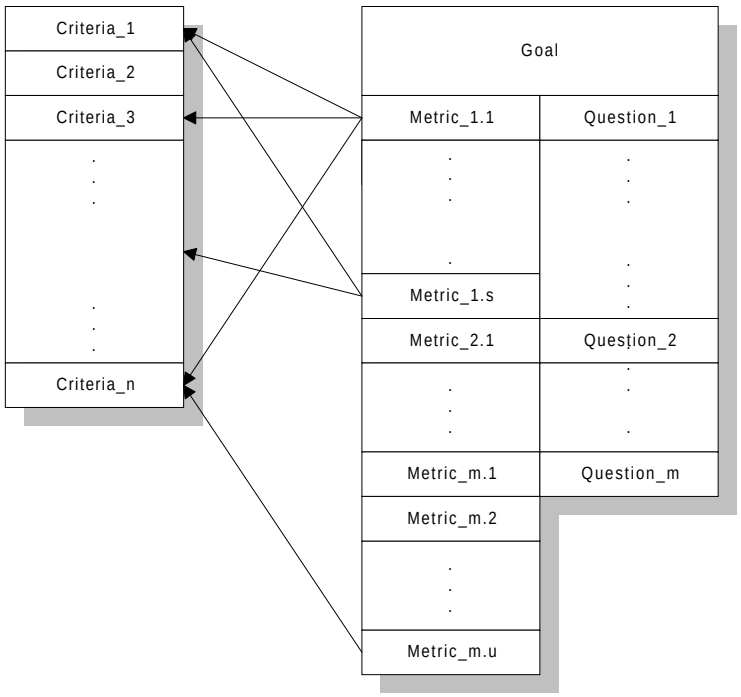

Figure 25. The tools evaluation framework.

The evaluation is based on defined criteria in terms of factors, such as performance, ease of use, tool integration/data exchange, robustness, functionality, and documentation. Three levels of evaluation were used: LOW, NORMAL or GOOD. LOW indicates that the tool does not have this function/property or is poor. NORMAL indicates that the tool has the function/property in question but it is hard to use and the results do not encourage continuous usage. GOOD means that the tool has this property/function and it is well supported.

The criteria themselves are not sufficient, since the justifications are vague, i.e. LOW, NORMAL or GOOD. Therefore the GQM approach was adopted to provide a more explicit justification for the tool evaluation. The results of a GQM analysis generally

consist of various values related to the criteria by a n:m relation. It means that the same metrics can justify several criteria and one criterion can use several metrics as its justification.

The criteria were used to design questions and metrics for the evaluation, which means that a list of questions and related metrics was created. Table 10 describes all the questions and their associated metrics. The metrics can be partly tool-specific, e.g. M3.1, which is specific to the AU toolset (AU), the reverse engineering tool ReverseNICE (RN), and the impact analysis toolset (IAS).

Table 10: A sample of the GQM-based questions and metrics.

Question	Metric	Definition
Q1		What do the tools require from the system?
	M1.1	Disk space required for the tool
	M1.2	Disk space required for additional tools
	M1.3	Resource usage
	M1.4	Memory usage
	M1.5	Number of available platforms
Q2		What do the tools require from the user during installation?
	M2.1	Number of parameters to be configured
Q3		How much time does the tool need for its functions?
AU	M3.1	Time to create code dependencies
AU	M3.2	Number of dependencies found
AU	M3.3	False dependencies found
AU	M3.4	Time to create HTML data from code
AU	M3.5	Number of HTML code files
AU	M3.6	Number of HTML code lines
RN	M3.1	Number of files
…		
IAS	M3.1	ReverseNICE TDIF (Transfer Data Interface) file size
…		
Q4		How does the tool support ease of use?
	M4.1	Input models value
	M4.2	Navigation value
	M4.3	Selection value
	M4.4	Component activation value
	M4.5	Application design principles value
	M4.6	Controls, groups and models value
…		

A sample of actual measurements in the AU toolset evaluation is shown in Table 11.

Table 11: A sample of the metrics collected from the AU toolset.

No	Metric	Value	Notes
M1.1	Disk space required for the tool	13.9 MB	
M1.2	Disk space required for additional tools	25 MB 1 MB	Webmaker Robochart
M1.3	Resource usage	1-2 MB during scripts, 1% CPU	Webmaker: 18-25 MB 35-82% CPU Robochart 2 MB
M1.4	Memory usage	1.3 MB	Webmaker 240 KB Robochart 684 KB
M1.5	Number of available platforms	1	SunOS, domain analysis part has Windows version
M2.1	Number of parameters to be configured	1	Assuming external tools are in path
M4.1	Input models value	3	Input is well supported for mouse. Focus is sometimes confusing.
M4.2	Navigation value	4	Navigation is well supported, some problems with menu functionalities.
M4.3	Selection value	2	There are only a few advanced features.
M4.4	Component activation value	1	No mnemonics or short-cuts.
M4.5	Application design principles value	1	Menus are not according to standards. The options are not shown via dialogue boxes.
M4.6	Controls, groups and models value	3	Generally according to standards.

Each metric was calculated, and the results were used to support the evaluation of the criteria. Each table indicates the metrics, results and possible additional notes. The notes column includes information about the separate AU tools used as a part of the AU toolset, such as Webmaker and Robochart. In addition,

metrics M3.1-M3.6 of the AU toolset were defined for each assessed software (Paper VII, Table 5).

The final results included the rating of six criteria on the scale LOW, NORMAL or GOOD. The following list is a sample of performance criteria, including subcriteria, such as response time, installation, start up, capacity, efficiency, and tool portability. Our evaluation produced the following rates:

Response Time: NORMAL

> The response time of the AU tools is considered satisfactory. Webmaker is the only tool, which requires a slightly longer execution time. A graphical interface for the AU tools was created with Tcl/Tk. Because Tcl/Tk is an interpretative language, the response times are slightly longer than with interfaces using interpretative languages.
>
> *Reference metrics:* M3.1 - M3.6

Installation: NORMAL

> Installation of the tools is straightforward because most of them are in the same directory. The AU tools do not check the system to locate third-party utilities. The separate tools can be installed by the system administrator or the user.
>
> *Reference metrics:* M1.1, M1.5, M2.1

Conclusions

The research and experiments showed that the tools evaluation is needed but it should be linked better to process where its is used. From the point of view of process improvement the problem was that companies should describe their software processes more explicitly for the requirements set in advance, such as the need for managing several modernisation cycles and for taking better into account of application-specific requirements.

ESA has defined a process model for space on-board software development, for example for change control, but it demands tailoring in each organisation. However, the tools evaluation provided practical results in the participating organisations.

6.2.2 The LEIVO project: Improvement of version control, software manufacturing and delivery

As the case studies in the other project included several parts, the role of the analysis of the current practices was important. The improvement of SCM and related elements was directed at version control, software manufacturing and order/delivery data management. Change management was beyond the scope, since these two SCM elements, version control and software manufacturing, were sufficient, taking into account the effort to be expended. In some of the case studies order/delivery data management was so essential that there was a need preliminarily to relate it to the SCM improvement.

Analysis of the current practices

The analysis adopted a qualitative approach providing a non-formal way of collecting information on current practices. The SCM framework nevertheless provided a context to analyse the separate SCM elements. In this phase of the SCM improvement the SCM elements of each company were related to the SCM maturity levels defining the state-of-the art in version control and software manufacturing. This phase of the LEIVO project also made results from all seven companies commensurable. The potential commercial tools were also analysed in parallel with this subphase, since they provided acceptable foundations for companywide SCM environments.

Definition of goals for improved practices

The goals for each company were very varied when defining goals of each company, because of the companies' backgrounds and their previous investments in SCM. The goals included the following items:

1. To evaluate the solution presented in Paper III and its possible utilisation in a new SCM environment.

2. To increase adaptability, relating it to the current environment, which implements SCM elements.

3. To improve the management of software products that do not belong to the deliverable software itself.

4. To develop the automated mass-customisation of software manufacturing.

5. To develop repeatable SCM, including version control and software manufacturing solutions.

The first goal dealt with a SCM environment totally based on commercial tools, e.g. PVCS, Polymake or MS-ACCESS (Figure 22). In addition to version control and software manufacturing aspects, the possible use of these tools required an analysis of usability, platform, security and reliability with regard to companies' current arrangements.

Improvement and its analysis

The results are shown in Table 12, where the dotted lines indicate the improvement of a SCM or related element in each company. Some companies concentrated entirely on improving their existing SCM environment by replacing an old solution with a new one. Therefore there is no need to use any dotted line indication in Table 12.

The SCM improvement can be assessed with regard to separate SCM-related elements. In most cases the companies had already adopted a commercial version control tool, although the version control solutions ranged from a low-level version control (level 0) to basic product management solutions (level 6) where version control also covered a large amount of documentation. In software manufacturing the companies concentrated at least on the definition and use of standardised makefile-builders under version control (level 3), although the common long-term goal was to create more flexible software manufacturing solutions (level 6). Although order/delivery management and customer data management were regarded as important, none companies had not yet the ability to improve this element.

Table 12: The results of the evaluation of SCM practices and their improvement.

	CM-related elements		
	SCM-specific elements		Logistics-specific process
Levels of improvement	**Version control**	**Software manufacturing**	**Order/delivery data management**
No product management **0**	Electronics company-1	Electronics company-1	Electronics company-1
Step 1			
Copying activity with manual versioning 1	Machine/Process automation company-1	Machine/Process automation company-3	
Step 2			
Basic version control **2**		Machine/Process automation company-3	Machine/Process automation company-1
Step 3			
Repeatable version control **3**	Electronics company-1 Machine/Process automation company-1 Telecommunication company-1 Machine/Process automation company-3	Electronics company-1	Telecommunication company-1 Machine/Process automation company-3 Telecommunication company-2
Step 4			
Basic configuration management **4**	Telecommunication company-1, 2 and 3 Machine/Process automation company-2	Telecommunication company-1, 2 and 3 Machine/Process automation company-2	Telecommunication company-1 and 3 Machine/Process automation company-2
Step 5			
Repeatable configuration management **5**			
Step 6			
Basic product management **6**	Telecommunication company-3	Telecommunication company-1	
Step 7			
Total product management **7-11**		Machine/Process automation company-3	Machine/Process automation company-3

Conclusions

This case project covered several companies and their SCM practices. This made improvement more complex. In this situation the SCM improvement levels and their SCM- and logistics-specific elements provided a way to discuss more focused about company-specific solutions.

As the common feature was to increase repeatability and adaptability of the SCM environments. This approach provided the possibility to focus more on version control and software manufacturing solutions.

The approach also showed the need to analyse separately CIs of the companies, which they have to store and relate to configurations. Since there was not yet final solutions with them, the discussion about product-level CIs and their relations with software configurations was not possible.

7. Introduction to papers

This chapter gives an introduction to the original publications included in this dissertation (Papers I - VII). They have all been accepted in scientific journals and volumes of conference proceedings within a four-year period. Paper I provides the motive for the research and emphasises the importance of software maintenance and related technical aspects. Papers II and VII describe technical solutions developed for software maintenance. Paper III presents a solution for SCM in the area of mechatronic systems, indicating at the same time the problems affecting software engineering in a situation in which there are other product technologies to be developed, and Paper IV considers process aspects as a part of the development of software maintenance. Paper IV also outlines the SCM improvement framework. Papers V and VI relate together various aspects arising in the previous papers, providing details of the incremental development steps in SCM and discussing experiences with the incremental approach. Paper VII provides results of quantitative measurements, particularly concerning tools for change control. The papers correspond to the research activities in the manner set out in Table 13.

Table 13: Relations between research activities and the original papers.

Research question	Description of the research activity	Paper(s)	Published in
Q1	Relate the role of software evolution and maintenance to SCM	I	1993
Q2	Investigate activities that can be distinguished in SCM	II, III	1993, 1994
Q3	Construct a SCM framework for developing software evolution and maintenance practices and provide evidence of its usefulness	V	1996
Q4	Investigate SCM improvement and conduct the first experiments to show its usefulness	IV, VI, VII	1995, 1996, 1998

7.1 Paper I, evaluation of software maintenance and its improvement

Paper I describes a CMM type evaluation of the role of software maintenance of embedded systems. The evaluation included 27 small and medium size companies whose products ranged from simple instruments to large telecommunication systems. The paper describes the evaluation method and shows, how the evaluation results can be analysed and packaged for future use. Packaging of the evaluation results was based on the work made by Oivo and Basili (Oivo and Basili 1992; Oivo 1994). The main points in Paper I are:

- to describe the nature of software maintenance in the case of embedded software,
- to show the importance of SCM in industry, and
- to relate the results to the improvement paradigm.

The work creates a basis for research into the roles of software maintenance and SCM and shows that software engineering requires specific forms of understanding, one of which is SCM.

7.2 Paper II, tool for software evolution and maintenance

Paper II provides a case in solving practical maintenance problems, when there is need to better understand the existing software. The background of this paper was to develop a reverse engineering tool for understanding of the old, weakly documented software. The input for the reverse engineering process is PL/M software. It is transformed to SA/SD-RT design models. The study concentrates on:

- outlining a solution by which it is possible to support software comprehension,
- to show the different code-level factors when developing more abstract descriptions of software that do not depend on the programming language used, and
- to acquire more information on the special characteristics of the maintenance of embedded software.

7.3 Paper III, a solution to the software configuration management problem

Paper III outlines a practical approach to the configuration and supply of machine control systems. A configuration system for the management and assembly of machine control software was built. This work emphasised the need for the incremental development of SCM practices. In addition to software, a need was found to consider other product technologies when developing the management and assembly of machine control software.

This paper describes:

- product technologies to be developed in parallel with software,
- logistics functions to be taken into account, and
- features of the assembly system.

7.4 Paper IV, process aspects of software maintenance

Paper IV concentrates on an on-board space application which acted as one of the cases for an application management environment called AMES. This paper describes the framework where the AMES environment was developed. The major elements of this paper consist of:

- a qualitative improvement framework,
- the analysis results of the application,
- the description of the AMES solution, and
- the description of a process model application management.

7.5 Paper V, levels of SCM practice improvement

Paper IV provides a basis for the incremental SCM improvement. This paper includes a first version and examples applying the improvement procedure. In this paper we relate both the mechatronics application presented in Paper III and the space application presented in Paper IV to the incremental SCM improvement.

The major aspects of this paper consist of:

- the description of the incremental SCM improvement, and
- the relation to the qualitative analysis as part of SCM improvement.

7.6 Paper VI, experiences of the improvement framework using qualitative analysis

Paper VI discusses the experiences, which have been obtained in co-operation with companies. The participating companies had very different SCM starting levels point to SCM ranging from simple manual version control to full-automatic CM systems with on-line change control. The use of commercial CM systems was also increasing, although companies had already used various basic version control systems in most cases. This provided a challenge and, on the other hand, the framework formed a suitable starting point for the analysis of each company.

The contents of this paper can be categorised into:

- the presentation of the improvement framework,
- the presentation of different elements related to CM, and
- the experiences provided by the companies.

7.7 Paper VII, experiences of the improvement framework using quantitative analysis

Paper VII provides the experiences, which have been obtained in the final evaluation of the tools of application management environment. The central element of the evaluation has been a new framework including evaluation criteria and Goal/Question/Metric (GQM)-based approach. They have produced detailed information about each tool. The paper describes:

- the tools evaluated,
- GQM-based evaluation framework, and
- sample of evaluation results.

The results are a part of a detailed report of the AMES project.

8. Conclusions and further research

The development of SCM is one of the solutions for problems concerning development and maintenance of more complicated embedded systems. However, SCM requires improvements. The problem is to find the right procedures to apply and further develop SCM in practice.

The goal of this research was to create an incremental approach for improving SCM support and practices in the case of embedded systems, taking into account the requirements imposed by evolution and maintenance of embedded systems. This research has analysed SCM producing a SCM framework, which was validated by industrial cases. In addition, the SCM framework provided a starting point for the further development of a SCM maturity model, which was also validated by industrial cases.

The SCM framework presented above was evaluated and applied in co-operation with several manufacturers of industrial embedded systems in the fields of mechatronics, space and other electronics applications. The main focus was on software-specific issues, although a preliminary consideration was also given of product and global data management.

The SCM framework consists of SCM-specific elements such as version control, software manufacturing, change control, and of SCM-related logistic processes. The maturity levels of this framework range from low-level version control to global product management including parallel solutions for SCM elements. Comparisons with some well-known approaches based on software process assessment, software process modelling and SCM practices provided indications of their positions in relation to SCM elements and SCM improvement. These approaches typically include SCM elements together with comprehensive solutions for process improvement, but they do not include such a specific description of SCM maturity levels in relation to individual SCM elements. In addition, the continuation to product level issues such as other technologies lies beyond the scope of this thesis.

Although the SCM concepts are in principle well-known, this research convinced the author of the difficulty of finding entirely unified concepts. Version control and software manufacturing are becoming well understood, but

the relation between release and change-oriented SCM is vague, as emerges when relating separate SCM items to each other. In addition, although there are already several problem tracking tools commercially available, the role of the database behind the problem tracking mechanism is not totally clear, since this database is related to both SCM and process improvement.

Companies have several computer-based design solutions which support product development, manufacturing and customer-related activities. The management of product level data is undergoing major changes at present, because companies are integrating the management of computer-based design solutions and software is one of the fields that can be integrated better at the product level. The highest level entails the global product aspects to be managed, including the virtual corporation, which the future product development business will have to take into account.

The approach adopted in this research followed a constructive model, including both conceptual and technical realisations. The industrial co-operation projects supported this approach, since the incremental development of SCM practices requires a conceptual view of each technical part, such as version control, release and change-oriented SCM parts. The technical realisations of version control and software manufacturing are based on commercial tools. Change-oriented SCM, excluding problem tracking databases, is still largely based on the precommercial solutions described in this research. The advanced commercial tool environment, such as ClearCase and Continuus, have produced their own solutions.

Although each company had specific SCM goals, the main target was standardised version control and software manufacturing practices, in spite of the fact that change control solutions will assume the main role in future SCM improvement. In release-oriented SCM the main interest will be in mass-customisation, where MIL-based descriptions will be used in assembly systems which require distributed solutions. The improvement of release production can be regarded as running parallel to change control. The companies have to resolve in which order improvements are to be made, as all of them are more expending effort and time in adopting and transferring old solutions to a new, advanced SCM environment. There are already commercial systems available, but the main problem in adopting may well be the expensive involved. It is also

essential for SME companies to adopt an incremental approach. In addition, there was a need to develop related processes, such as logistic and software resolution-specific ones, and this makes improvement quite complicated.

Improvement of the SCM process also requires a software process improvement (SPI) procedure of its own. In our case we adopted an inductive approach. This procedure proved to be beneficial, since the development of specific software processes such as SCM calls for concentration on SCM-specific features in particular. In addition, process improvement will be one of the essential development targets in software engineering in the future. Thus, each process, including SCM, requires a specific analysis, as demonstrated in this research. Different processes such as SCM requires their own maturity steps, and these are examined in this thesis. SCM forms one of the processes to be improved in the PR^{2}IMER concept. PR^{2}IMER provides a way to proceed in the improvement process, but SCM-specific features have to be better understood and related to each other. Our results will be transferred to part of the service package SCM-PR^{2}IMER, which will form a basis for industrial co-operation when developing companywide SCM solutions.

The work presented in this thesis has produced the following new research initiatives for the future:

- *Increased SCM usage*: The development of SCM related software technology to other product technologies. Experiences from research and industrial work up to now have been used to develop software and other technologies for individual CM solutions. This will lead to difficulties in the future, as companies want to produce more integrated products.
- *Distributed SCM:* New business solutions such as the virtual corporation will also place new requirements on software engineering. This activity will increase because of more use of subcontracting and because of better communication mechanisms. A new EU/ESPRIT project VISCOUNT is being launched partly on the basis of ideas presented in this paper to develop distributed CM support.
- *SCM as a support for distributed product solutions:* Since mass-customisation will characterise much of software manufacturing and the solutions become more distributed, MIL languages will be increasingly used. Commercial SCM systems will have to consider distribution of

embedded systems, and MIL technology will provide a worthy alternative solution for it.

- *SPI for SCM:* The SCM improvement procedure will become part of the SCM-PR^{2}IMER service package. In addition to problems in understanding the technical dependencies between different SCM elements, it is difficult to understand the investments, effort and time required to achieve the desired level. The procedure presented in this thesis can be regarded as an initial way of visualising and assessing each SCM improvement activity. One of the goals of the VISCOUNT project will be to evolve the SCM-PR^{2}IMER concept to meet future industrial improvement requirements.

References

Adamson, M. 1990. Small Real-Time Design: from Microcontrollers to RISC Processors. Sigma Press, Wilmslow, UK. 191 p.

Agresti, W. 1986. What Are the New Paradigms? In: Agresti, W. (ed.), New Paradigms for Software Development. IEEE Computer Society Press, Washington, D.C. Pp. 6 - 10.

Alonso, A. and de la Puente, J.A. 1993. Dynamic Replacement of Software in Hard Real-Time Systems. In: Proceedings on the Fifth Euromicro Workshop on Real-Time Systems, Oulu, Finland, June 1993. IEEE Computer Society Press, Los Alamitos, California. Pp. 76 - 81.

Ambriola, V., Bendix, L. and Ciancarini, P. 1990. The Evolution of Configuration Management and Version Control. Software Engineering Journal, Vol. 5, No. 6, pp. 303 - 310.

AMES 1994a. AMES - Methodology and Process Model. AMES Participants, Cap Gemini Innovation, Grenoble, France. Esprit 3 Project 8156, AMES Deliverable D111, version 2.0, May 1994. 93 p.

AMES 1994b. AMES - Evaluation Scenarios. AMES Participants, Cap Gemini Innovation, Grenoble, France. Esprit 3 Project 8156 AMES Deliverable D42, version 1.0, December 1994. 101 p.

AMES 1995. AMES - Application Management Environments and Support. Cap Gemini Innovation, Grenoble, France. Esprit 3 Project 8156, Technical Annex, Version 3.0, November 1995. 85 p.

Arthur, L. 1988. Software Evolution: The Software Maintenance Challenge. John Wiley & Sons, New York. 254 p.

Aumaitre, J.-M., Harjani, D.-R. and Dowson, M. 1993. PROMESSE - Final Report, PROMESSE Project ESA 10081/92/NL presented in section 3.2.1 JG(SC). 51 p.

Barros, S., Bodhuin, Th., Escudié, A., Queille, J.-P. and Voidrot, J.F. 1995. Supporting Impact Analysis: a Semi-Automated Technique and Associated Tool. In: Caldiera G. and Bennett, K. (eds.), Proceedings of the International Conference on Software Maintenance (ICSM'95), Opio (Nice), France, October 1995. IEEE Computer Society Press, Los Alamitos, California. Pp. 52 - 61.

Battaglia M. and Savoia, G. 1997. ReverseNICE: A Nice Way to Reengineer Legacy Systems. In: T.-D. Guyenne (ed.), Proceedings of the 2nd International Conference on Data Systems in Aerospace (DASIA'97), Seville, Spain, May 1997. European Space Agency (ESA) SP-409, Noordwijk, the Netherlands. Pp. 195 - 201.

Belady, L.A. and Lehman, M.A. 1976. A Model of Large Program Development. IBM Systems Journal, Vol. 15, No. 3, pp. 225 - 252.

Belkhatir, N., Estublier, J. and Melo. W. 1994. ADELE-TEMPO: An Environment to Support Process Modelling and Enaction. In: Finkelstein, A., Kramer, J. and Nuseibeh, B. (eds.), Software Process Modelling and Technology. Research Studies Press Ltd, Taunton, Somerset, UK. Pp. 187 - 222.

Bell 1994, TRILLIUM - Telecom Product Development Process Capability, Version 3.0. Bell Canada, December 1994. 116 p.

Bennett, K. 1993. An Overview of Maintenance and Reverse Engineering. In: van Zuylen, H. (ed.), The REDO Handbook - A Compendium of Reverse Engineering for Software Maintenance. John Wiley & Sons, Chichester, UK, Chapter 2. Pp. 43 - 60.

Bennett, K., Cornelius, B., Munro, M. and Robson, D. 1991. Software Maintenance. In: McDermid, J.A. (ed.), Software Engineer's Reference Book, Butterworth-Heinemann, Oxford, UK. Chapter 20. 18 p.

Berlack, R. 1992. Software Configuration Management. John Wiley & Sons, New York. 346 p.

Bershoff, E.H., Henderson, V.D. and Siegel., S.G. 1979. Principles of Software Configuration Management. Prentice-Hall, Englewood Cliffs, New Jersey. 385 p.

Bishop, J.M. 1994. Languages for Configuration Programming: a Comparison. University of Pretoria, South Africa. Technical Report 94/04. 27 p.

Boehm, B.W. 1976. Software Engineering. IEEE Transactions on Computing, Vol. 2, pp. 1226 - 1242.

Boehm, B.W. 1988. A Spiral Model of Software Development and Enhancement. IEEE Computer, Vol. 21, No. 5, pp. 61 - 72.

Boldyreff, C., Burd, E.L. and Hather, R.M. 1994. An Evaluation of the State of the Art for Application Management. In: Muller H.A. and Georges, M. (eds.), Proceedings of the International Conference on Software Maintenance (ICSM'94), Victoria, Canada, September 1994. IEEE Computer Society Press, Los Alamitos, California. Pp. 161 - 169.

Boldyreff, C., Burd, E.L., Hather, R.M., Mortimer, R.E., Munro, M. and Younger, E.J. 1995. The AMES Approach to Application Understanding: a case study. In: Caldiera G. and Bennett, K. (eds.), Proceedings of the International Conference on Software Maintenance (ICSM'95), Opio (Nice), France, October 1995. IEEE Computer Society Press, Los Alamitos, California. Pp. 182 - 191.

Boldyreff, B., Newman, J. and Taramaa, J. 1996. Managing Process Improvement in Virtual Software Corporations. In: Proceedings of the Fifth Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE), Stanford, California, June 1996. IEEE Computer Society Press, Los Alamitos, California. Pp. 142 - 147.

Brown, A.W., Christie, A.M. and Dart, S.A. 1995. An Examination of Software Maintenance Practices in a U.S. Government Organization. Journal of Software Maintenance, Vol. 7, No. 4, pp. 223 - 238.

Buckley, F.J. 1996. Implementing Configuration Management - Hardware, Software and Firmware, Second Edition. IEEE Computer Society Press, Los Alamitos, California. 380 p.

Buschman, F., Meunier, R., Rohnert, H., Sommerlad, P and Stal, M. 1996. Pattern-oriented Software Architecture, a System of Patterns. John Wiley & Sons, Chichester, UK. 457 p.

Cagan, M. 1994. Untangling Configuration Management - Mechanism and Methodology in SCM Systems. In: Estublier, J. (ed.), Software Configuration Management: ICSE SCM-4 and SCM-5 Workshops Selected Papers, Lecture Notes 1005 in Computer Science, International Workshop on Software Configuration Management (SCM4), Baltimore, Maryland, May 1994. Springer-Verlag, Heidelberg, Germany. Pp. 35 - 52.

Capretz, M.A.M. and Munro, M. 1992. COMFORM - A Software Maintenance Method Based on the Software Configuration Management Discipline. In: Kellner, M. (ed.), Proceedings of the International Conference on Software Maintenance (ICSM'92), Orlando, Florida, November 1992. IEEE Computer Society Press, Los Alamitos, California. Pp. 183 - 192.

Chapin, N. 1989. Changes in Change Control. In: Proceedings of the International Conference on Software Maintenance (ICSM'89), Miami, Florida, October 1989. IEEE Computer Society Press, Los Alamitos, California. Pp. 246 - 253.

Chikofsky, E.J. and Cross, J.H. 1990. Reverse Engineering and Design Recovery: A Taxonomy. IEEE Software, Vol. 7, No. 1, pp. 13 - 17.

Christie, A.M. 1994. A Practical Guide to the Technology and Adoption of Software Process Automation. Carnegie-Mellon University, Pittsburgh, Pennsylvania. Technical Report CMU/SEI-94-007. 128 p.

CIMdata Inc. 1996. What Is Product Data Management?
http://www.std.com/Newbury/CIMdata/pages/what_is_pdm.htm. 2 p.

Coene, Y. 1992. A Generic Model for ESSDE Software Configuration Management. ESSDE/STREP/A21, ESSDE Reference Facility Project, ESA 8900/90/NL/US(SC). 94 p.

Conradi, R., Fernström, C. and Fuggetta, A. 1994. Concepts for Evolving Software Processes. In: Finkelstein, A., Kramer, J. and Nuseibeh, B. (eds.), Software Process Modelling and Technology. John Wiley & Sons Inc., Chichester, UK. Pp. 9 - 31.

Conradi, R. and Westfechtel, B. 1998. Version Models for Software Configuration Management. ACM Computing Surveys, Vol. 30, No. 2, pp. 232 - 282.

Dart, S.A. 1990. Spectrum of Functionality in Configuration Management Systems. Carnegie-Mellon University, Pittsburgh, Pennsylvania. Technical Report CMU/SEI-90-11. 38 p.

Dart, S.A. 1991. Concepts in Configuration Management Systems. In: Feiler, P.H. (ed.), Proceedings of the 3rd International Workshop on Software Configuration Management (SCM3), Trondheim, Norway, June 1991. ACM Press, Baltimore, Maryland. Pp. 1 - 18.

Dart, S.A. 1992a. Past, Present and Future of CM Systems. Carnegie Mellon University, Pittsburgh, Pennsylvania. Technical Report CMU/SEI-92-8. 28 p.

Dart, S.A. 1992b. Parallels in Computer-Aided Design Framework and Software Development Environments Efforts. In: Newman, M. and Rhyne, T. (eds.), Proceedings of the Third IFIP WG 10.2/WG 10.5 Workshop on Electronic Design Automation Frameworks, Bad Lippspringe, Germany, March 1992. North Holland, Amsterdam, the Netherlands. Pp. 175 - 189.

Dart, S.A. 1995. Adopting an Automated Configuration Management Solution. In: the Eight International Workshop on Computer-Aided Software Engineering (CASE'95), Toronto, Ontario, July 1995, an invited speech. 15 p.

Dart, S.A. 1996. Best Practice for a Configuration Management Solution. In: Sommerville, I. (ed.), Software Configuration Management ICSE SCM-6

Workshop Selected Papers, Lecture Notes 1167 in Computer Science, Proceedings of the 6th International Workshop on Software Configuration Management (SCM6), Berlin, Germany, March 1996. Springer-Verlag, Heidelberg, Germany. Pp. 239 - 255.

Davidow, W.H. and Malone, M.S. 1992. The Virtual Corporation: Structuring and Revitalizing the Corporation for the 21st Century. Harper Business, New York. 294 p.

Davis, A.M. 1990. Software Requirements - Analysis & Specification. Prentice-Hall, Englewood Cliffs, New Jersey. 512 p.

Davis, A.M. and Bershoff, E.H. 1991. Impacts of Lifecycle Models on Software Configuration Management. Communications of the ACM, Vol. 34, No. 8, pp. 104 - 118.

Deitz, D. 1990. Pulling The Data Together. Mechanical Engineering, Vol. 112, No. 2, pp. 56 - 57.

De Micheli, D. 1996. Hardware Software Co-Design: Application Domains and Design Technologies. In: De Micheli G. and Sami, M. (eds.), Hardware/Software Co-Design, NATO ASI Series, Vol. 210. Kluwer Academic Publishers, Dordrecht, the Netherlands. Pp. 1 - 28.

Deming, W.E. 1982. Out of the Crisis. MIT Centre for Advanced Engineering Study, Cambridge, MA.

DeRemer, F. and Kron, H.H. 1976. Programming-in-the-Large Versus Programming-in-the-Small. IEEE Transactions on Software Engineering, Vol. 2, No. 2, pp. 80 - 86.

DoD 1988. DoD-STD-2167A, Defense System Software Development, Department of Defense, February 1988.

Doize, M.-S., Voidrot, J.-F. and Escudie, A. 1994. Traceability Data Models. AMES Project Deliverable D213, version 2.0, Grenoble, France: Cap Gemini Innovation and Matra Marconi Space, November 1994. 75 p.

ESA 1989. PSS-01-11 issue 1, Configuration Management and Control for ESA Space Systems. European Space Agency, Noordwijk, the Netherlands. 93 p.

ESA. 1991. PSS-05-0 issue 2. ESA Software Engineering Standards, Issue 2. European Space Agency, Noordwijk, the Netherlands. 130 p.

Estublier, J. and Casallas, R. 1994. The Adele Configuration Manager. In: Tichy, W.F. (ed.), Configuration Management. John Wiley & Sons, Chichester, UK. Pp. 99 - 133.

Estublier, J., Favre, J-M. and Morat, P. 1998. Toward SCM / PDM integration? In: Magnusson, B. (Ed.), System Configuration Management, Lecture Notes 1439 in Computer Science, International Symposium on System Configuration Management (SCM8), Brussels, Belgium, July 1998. Springer-Verlag, Heidelberg, Germany. Pp. 75 - 94.

Favaro, J., Coene, Y. and Casucci, M. 1994. The European Space Software Development Environment Reference Facility Project: A Status Report. ACM SIGSOFT, Software Engineering Notes, Vol. 19, No. 2, pp. 68 - 71.

Feiler, P. and Downey, G. 1990. Transaction-Oriented Configuration Management. Carnegie Mellon University, Pittsburgh, Pennsylvania. Technical Report CMU/SEI-90-7. 60 p.

Feiler, P. H. 1991. Configuration Management Models in Commercial Environments. Carnegie Mellon University, Pittsburgh, Pennsylvania. Technical Report CMU/SEI-91-7. 54 p.

Feldman, S.I. 1979. Make - A Program for Maintaining Computer Programs. Software - Practice and Experience, Vol. 9, No. 3, pp. 255 - 265.

Fernström, C. 1993. ProcessWEAVER: Adding Process Support to UNIX. In: Osterweil L. (ed.), Proceedings of the 2nd International Conference on the Software Process, Berlin, Germany, February 1993. IEEE Computer Society Press, Los Alamitos, California. Pp. 28 - 40.

Flynn, J.M. 1995. MAKEing Projects. Embedded Systems Programming, Vol. 8, No. 9, pp. 42 - 65.

Forte, G. 1993. Software Configuration Management. CASE OUTLOOK, Vol. 7, No. 2, pp. 3 - 45.

Foster, J.F. 1993. Cost Factors in Software Maintenance. University of Durham, School of Engineering and Computer Science, UK. Ph.D. Thesis. 138 p.

Foster, J.R., Jolly, A.E.P. and Norris, T. 1989. An Overview of Software Maintenance. British Telecom Technology Journal, Vol. 7, No. 4, pp. 37 - 46.

Frakes, W.B. and Isoda, S. 1994. Success Factors of Systematic Reuse. IEEE Software, Vol. 11, No. 9, pp. 15 - 19.

Frieder, O., Herman, G.E., Mansfield, W.H. and Segal, M.E. 1989. Dynamic Program Modification in Telecommunication Systems. In: Proceedings of the Seventh International Conference on Software Engineering for Telecommunication Switching Systems (SETSS89), Bournemouth, UK, July 1989. IEE, London, UK. Pp. 168 - 172.

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

Gomaa, H. 1993. A Reuse-Oriented Approach for Structuring and Configuring Distributed Applications. Software Engineering Journal, Vol. 8, No. 2, pp. 61 - 71.

Grundmann, W. 1997. Flash Memory Technology and Techniques. In: Proceedings of the Embedded Systems Conference, September 1997, Santa Jose, California, Volume 1. Miller Freeman Inc., San Francisco, California. Pp. 393 - 414.

Gulla, B., Martinsen, S.-A. and Dahlsengen E. 1993. Survey of SCM and Make tools. SINTEF, Trondheim, Norway. PROTEUS Project Technical Report P-WP-A223-BG001-SIN.4, January 1993. 34 p.

Habermann, A.N. and Notkin, D. 1986. Gandalf: Software Development Environments. IEEE Transactions on Software Engineering, Vol. 12, No. 12, pp. 1117 - 1127.

Hakkarainen, K. Kemppainen, P. and Karjalainen, A. 1988. A Knowledge-Based Configurer for Embedded Computer Systems Software - Real Life Experience. In: Proceedings of the International Workshop on Artificial Intelligence for Industrial Application, Hitachi City, Japan, May 1988. IEEE Computer Society Press, Washington, D.C. Pp. 608 - 613.

Harjani, D-R. and Queille, J-P. 1992. A Process Model for the Maintenance of Large Space Systems Software. In: Proceedings of the International Conference on Software Maintenance (ICSM'92), Orlando, Florida, November 1992. IEEE Computer Society Press, Los Alamitos, California. Pp. 127 - 136.

Harjani, D.-R. 1993. PROMESSE - ESA Software Development Process Models Specification. PROMESSE Project ESA 10081/92/NL/JG(SC). 181 p.

Harjani, D.-R., Queille, J.-P., Taramaa, J. and Ketola, T. 1995a. AMES Methodology and Process Model for the Aerospace Domain. Esprit 3 Project 8156 AMES Deliverable D1.2.1, April 1995. 51 p.

Harjani, D.-R., Martelli, A. and Aquilino, D. 1995b. Lessons Learnt from Putting Process Modelling Techniques into Practice. In: Proceedings of Software. 5 p.

Hauptmann, S. and Wasel, J. 1996. On-line Maintenance with On-the-fly Software Replacement. In: Proceedings of the 3rd International Conference on Configurable Distributed Systems (CDS96), Annapolis, Maryland, May 1996. IEEE Computer Society Press, Los Alamitos, California. Pp. 70 - 80.

Heikkinen, M. 1997. A Concurrent Engineering Process for Embedded Systems Development. VTT Publications 313. Espoo, Finland: Technical Research Centre of Finland (VTT). Licentiate Thesis. 93 p.

Heusala, H. and Tiensyrjä, K. 1995. Covalidation Approaches According to Embedded System Classification. In: Computer-Aided Validation Engineering (CAVE'95) Workshop, Kenmare, Ireland, December 1995. 3 p.

van der Hoek, A., Heimbigner, D. and Wolf, A.L. 1996. A Generic, Peer-to-Peer Repository for Distributed Configuration Management. In: Proceedings of the 18th International Conference on Software Engineering (ICSE18), Berlin, Germany, March 1996. IEEE Computer Society, Los Alamitos, California. Pp. 308 - 317.

Honka, H. 1990. A Simulation-based Approach to Testing Embedded Software. VTT Publications 124. Espoo, Finland: Technical Research Centre of Finland (VTT). Licentiate Thesis. 118 p.

Humphrey, W.S. 1988. Characterizing the Software Process: A Maturity Framework. IEEE Software, Vol. 5, No. 3, pp. 73 - 79.

Hunt, J.J., Vo. K.-P. and Tichy, W.F. 1996. An Empirical Study of Delta Algorithms. In: Sommerville, I. (ed.), Software Configuration Management ICSE SCM-6 Workshop Selected Papers, Lecture Notes 1167 in Computer Science, Proceedings of the 6th International Workshop on Software Configuration Management (SCM6), Berlin, Germany, March 1996. Springer-Verlag, Heidelberg, Germany. Pp. 49 - 66.

Hurst, W. and Dennis, J. 1996. Report on the Major Issues and Concerns of Industry Group Associations and Their Members on the Future Use of Embedded Microprocessors with Their Respective Industries - A report for the OMI Software Architecture Forum, July 1996, Draft, 53 p, in the www address: http://www.osaf.org/library/issues.pdf.

IEEE 1042-1987. IEEE Guide to Software Configuration Management Plans (ANSI). The Institute of Electrical and Electronics Engineers, Piscataway, New Jersey. 92 p.

IEEE 610.12-1990. IEEE Standard Glossary of Software Engineering Terminology (ANSI). The Institute of Electrical and Electronics Engineers, Piscataway, New Jersey. 83 p.

IEEE 828-1990. IEEE Standard for Software Configuration Management Plans (ANSI). The Institute of Electrical and Electronics Engineers, Piscataway, New Jersey. 16 p.

IEEE 1219-1992. IEEE Standard for Software Maintenance (ANSI). The Institute of Electrical and Electronics Engineers, Piscataway, New Jersey. 39 p.

Iivari, J. 1991. A Paradigmatic Analysis of Contemporary Schools of IS Development. The European Journal of Information Systems, Vol. 1, No. 4, pp. 249 - 272.

ISO 9000-3. 1991. Quality Management and Quality Assurance Standards - Part 3: Guidelines for the Application of ISO 9001 on the Development, Supply and Maintenance of Software. ISO/IEC Copyright Office, Geneva, Switzerland. 13 p.

ISO 10303-1. 1994. Industrial Automation Systems and Integration - Product Data Representation and Exchange - Part 1: Overview and Fundamental Principles. ISO/IEC Copyright Office, Geneva, Switzerland. 17 p.

ISO 10303-11. 1994. Industrial Automation Systems and Integration - Product Data Representation and Exchange - Part 11: Description Methods: The EXPRESS Language Reference Manual. ISO/IEC Copyright Office, Geneva, Switzerland. 208 p.

ISO/IEC 12207. 1995. Informational Technology - Software Life Cycle Processes, International Standard. ISO/IEC Copyright Office, Geneva, Switzerland. 57 p.

ISO/IEC 15504. 1995. SPICE - Software Process Assessment - Part 5: Construction, Selection and Use of Assessment Instruments and Tools, Version 1.00. ISO/IEC Copyright Office, Geneva, Switzerland. 130 p.

Juopperi, J., Lehtola, A., Pihlajamaa, O., Sladek, A. and Veijalainen, J. 1995. Usability of Some Workflow Products in an Inter-organizational Setting. In: Proceedings of IFIP WG8.1 Working Conference on Information Systems for Decentralized Organizations, Trondheim, Norway, August 1995. Chapman & Hall, London, UK. Pp. 81 - 95.

Järvinen, P. and Järvinen, A. 1995. On Methods in Research Work. Tampere, Finland: Opinpaja. 140 p. (in Finnish)

Karjalainen, J. 1987. A Classification Scheme for Embedded Computer Systems. In: Proceedings of the 14th Annual Conference of IEEE Industrial Electronics Society (IECON'88), October 198, Singapore. IEEE Society Press, Washington, D.C. Pp. 427 - 435.

Karjalainen, J., Mäkäräinen, M., Komi-Sirviö, S. and Seppänen, V. 1996. Practical Process Improvement for Embedded Real-Time Software. The Journal of Quality Engineering, Vol. 8, No. 4, pp. 565 - 573.

Kemppainen, P. 1986. An Environment for High-Volume Production of Embedded Systems Software. Department of Electrical Engineering, University of Oulu, Finland. Licentiate Thesis. 75 p. + app. 20 p.

Kilpi, T. 1996. Evaluating the Maturity of Software Product Management: A Case Study in Three Companies. In: Proceedings of the Ninth Australian Software Engineering Conference (ASWEC'96), July 1996, IREE Society. 9 p.

Kooshian, S. 1994. Managing Software Teams in a Hardware-Oriented Company. In: Proceedings of the Sixth Annual Embedded Systems Conference, September 1994, Santa Clara, California, Volume 1. Miller Freeman Inc., San Francisco, California. Pp. 495 - 501.

Krogstie, J. and Sölvberg, A. 1994. Software Maintenance in Norway: A Survey Investigation. In: Muller H.A. and Georges, M. (eds.), Proceedings of the International Conference on Software Maintenance (ICSM'94), September 1994, Victoria, Canada. IEEE Computer Society Press, Los Alamitos, California. Pp. 304 - 313.

Kuvaja, P., Similä, J., Kraznik L., Bicego A., Saukkonen S. and Koch, G. 1994. Bootstrap: Europe's Assessment Method. Blackwell, Oxford, UK. 149 p.

Laitinen, K. 1995. Natural Naming in Software Development and Maintenance. VTT Publications 243. Espoo, Finland: Technical Research Centre of Finland (VTT). Ph.D. Thesis. 99 p. + app. 70 p.

Laitinen, K., Taramaa, J., Heikkilä, M. and Rowe, N.C. 1997. Enhancing Maintainability of Source Programs through Disabbreviation. The Journal of Systems and Software, Vol. 17, No. 5, pp. 117 - 128.

Latvakoski, J. 1997. Integration Test Automation of Embedded Communication Software. VTT Publications 318, Espoo, Finland: Technical Research Centre of Finland (VTT). Licentiate Thesis. 98 p. + app. 28 p.

Lehman, M. 1980. Program, Life-cycles and Laws of Software Evolution. Proceedings of IEEE, Vol. 68, No. 9, pp. 1060 - 1076.

Lehtinen, K. 1994. Software Configuration Management for Embedded Software in Mobile Phone. Department of Information Technology, Lappeenranta University of Technology, Finland. Master‘s Thesis. 64 p + 21 app. (in Finnish)

Leveson, N. G. 1986. Software Safety: Why, What and How? ACM Computing Surveys, Vol. 18, No. 2, pp. 125 - 163.

Liddiard, J. 1994. An Introduction to Safety Critical Systems. Embedded System Engineering, Vol. 2, No. 2, pp. 38 - 42.

Lientz, B. and Swanson, S. 1980. Software Maintenance Management - A Study of the Maintenance of Computer Application Software in 487 Data Processing Organizations. Addison-Wesley Publishing Company, Reading, Massachusetts. 213 p.

Lopéz, P. and Rodríguez, A.I. 1995. Patching On-Board Ada Software. In: T.-D. Guyenne (ed.), Proceedings of an International Symposium on On-Board Real-Time Software (ISOBRTS’95), Noordwijk, the Netherlands, November 1995.

European Space Agency (ESA) SP-375, Noordwijk, the Netherlands. Pp. 69 - 75.

Lyon, D.D. 1997. Practical CM, Draft Revision L of Publication. Raven Publishing Company, Pittsfield, MA. 301 p.

MacKay, S.A. 1995. The State of the Art in Concurrent, Distributed Configuration Management. In: Estublier, J. (ed.), Software Configuration Management: ICSE SCM-4 and SCM-5 Workshops Selected Papers, Lecture Notes 1005 in Computer Science, International Workshop on Software Configuration Management (SCM5), Seattle, Washington, April 1995. Springer-Verlag, Heidelberg, Germany. Pp. 180 - 193.

Madhavji, N.H. 1991. The Process Cycle. Software Engineering Journal, Vol. 6, No. 5, pp. 234 - 242.

Mittag, L. 1996. Trends in Hardware/Software Codesign. Embedded Systems Programming, Vol. 9, No. 1, pp. 36 - 45.

Mortensen, U. 1995. Opening Address. In: T.-D. Guyenne (ed.), Proceedings of an International Symposium on On-Board Real-Time Software (ISOBRTS'95), Noordwijk, the Netherlands, November 1995. European Space Agency (ESA) SP-375, Noordwijk, the Netherlands. Pp. 7 - 8.

Mukari, T. 1988. Software Production for Cellular Mobile Phone Systems. Department of Electrical Engineering, University of Oulu, Finland. Licentiate Thesis. 64 p. + app. 15 p.

Murrach, W. 1998. Configuration Control for Embedded System, In: Proceedings of the 3rd International Conference on Data Systems in Aerospace (DASIA'98), Athens, Greece, May 1998. European Space Agency (ESA) SP-422, Noordwijk, the Netherlands. Pp. 223 - 229.

Mäkäräinen, M. and Taramaa, J. 1995. Quality Aspects in Evolution of Real-Time Embedded Software. In: Ross. M., Brebbia, C.A., Staples, G. and Stapleton, J. (eds.), Quality Management. Software Quality Management III Vol. 1, Proceedings of the 3rd International Conference on Software Quality

Management, Seville, Spain, April 1995. Computational Mechanics Publications, Southampton, UK. Pp. 313 - 320.

Mäkäräinen, M. 1996. Application Management Requirements of Embedded Software. VTT Publications 286, Espoo, Finland: Technical Research Centre of Finland (VTT). Licentiate Thesis. 99 p.

Niemelä, E., Taramaa, J. and Seppänen, V. 1995. Integration of Prototyping and Target System Development for Embedded Machine Control Software. In: Hamza, M.H. (ed.), Proceedings of the 14th IASTED International Conference, IGLS, Austria, February 1995. International Association of Science and Technology for Development - IASTED, Anaheim, California. Pp. 12 - 17.

Noll, J. and Scacchi, W. 1997. Supporting Distributed Configuration Management in Virtual Enterprises. In: Conradi, R. (ed.), Software Configuration Management: ICSE'97 SCM-7 Workshop, Lecture Notes 1235 in Computer Science, International Workshop on Software Configuration Management (SCM7), Boston, Massachusetts, May 1997. Springer-Verlag, Heidelberg, Germany. Pp. 142 - 160.

Nummelin, G. 1998. Succeeded Implementation of PDM System. In: PDM Summit'98 Seminar, Espoo, Finland, June 1998. IIR Finland, Helsinki. Handouts. 26 p (in Finnish).

Oivo, M. and Basili, V.R. 1992. Representing Software Engineering models: The TAME Goal Oriented Approach. IEEE Transactions on Software Engineering, Vol. 14, No. 6, pp. 758 - 773.

Oivo, M. 1994. Quantitative Management of Software Production Using Object-oriented Models. VTT Publications 169, Espoo, Finland: Technical Research Centre of Finland (VTT). Ph.D. Thesis. 72 p. + app. 67 p.

Oman, P., Hagemeister, J. and Ash, D. 1992. A Definition and Taxonomy for Software Maintainability. University of Idaho, Software Engineering Lab, Idaho. Technical Report 91-08. 30 p.

Osterweil, L. 1987. Software Processes Are Software Too. In: Proceedings of the 9th International Conference on Software Engineering (ICSE9), Monterey, California, March 1987. IEEE Computer Society Press, Los Alamitos, California. Pp. 2 - 13.

Paulk, M.C., Weber, C.V., Curtis, B. and Chrissis, M.B. 1995. The Capability Maturity Model: Guidelines for Improving the Software Process. Addison-Wesley, Reading, Massachusetts. The SEI Series in Software Engineering. 441 p.

Pressman, R.S. 1996. Software Engineering - A Practitioner's Approach, Forth Edition. McGraw-Hill, New York. 852 p.

Prieto-Diaz, R. and Neighbours, J. 1986. Module Interconnection Languages. Journal of Systems and Software, Vol. 6, No. 4, pp. 307 - 334.

Pulli, P. 1991. A Interpreter for Heterogeneous Prototypes. VTT Publications 79, Espoo, Finland: Technical Research Centre of Finland (VTT). Ph.D. Thesis. 46 p. + app. 76 p.

Pulli, P. and Heikkinen, M. 1993. Concurrent Engineering for Real-Time Systems. IEEE Software, Vol. 10, No. 11, pp. 39 - 44.

Purtilo, J.M. 1994. The Polylith Software Bus. ACM Transactions on Programming Languages and Systems, Vol. 16, No. 1, pp. 151 - 174.

Rigg, W., Burrows, C. and Ingram, P. 1995. Ovum Evaluates: Configuration Management Tools, OVUM report. OVUM, London, UK. 364 p.

Rochkind, M.J. 1975. The Source Code Control System. IEEE Transactions on Software Engineering, Vol. 1, No. 4, pp. 364 - 370.

Rosch, W.L. 1994. The Hardware Bible, Third Edition. Sams Publishing, Indianapolis, Indiana. 1202 p.

Rytilä, H. 1994. Reverse Engineering Technology as a Tool of Embedded Software - A solution from PL/M code to Structure Charts. VTT Research

Notes 1593. Espoo, Finland: Technical Research Centre of Finland (VTT). Master's Thesis. 84 p. + app. 16 p.

Savola, R., Pulli, P., Heikkinen, M., Seppänen, V., Kurki, M. and Taramaa, J. 1995. Concurrent Development of Multi-Technology Products - Integrating Virtual and Executable Models. In: Sobolewski, A.J. (ed.), Proceedings of the Concurrent Engineering 95 (CE95), Johnstown, Pennsylvania, June 1995. Concurrent Technologies Corporation, Oakland, California. Pp. 645 - 651.

Segal, M.E. and Frieder, O. 1993. On-the-Fly Program Modification: Systems for Dynamic Updating. IEEE Software, Vol. 10, No. 2, pp. 53 - 65.

Seppänen, V. 1990. Acquisition and Reuse of Knowledge to Design Embedded Software. VTT Publications 66, Espoo, Finland: Technical Research Centre of Finland (VTT). Ph.D. Thesis. 218 p. + app. 10 p.

Seppänen, V., Matsumoto, Y. and Pesonen, P. 1993. Conceptual Modeling of Families of Real-Time Systems. In: Proceedings of the 5th Euromicro Workshop on Real-Time Systems, Oulu, Finland, June 1993. IEEE Computer Society Press, Los Alamitos, California. Pp. 14 - 19.

Seppänen, V. Tuomikoski, T., Hintikka, P., Seppänen, P. and Väänänen, K. 1994. Embedded Software Engineering Megatrends, Technical Report of the SULAKO project. VTT Electronics, Oulu, Finland. 47 p.

Seppänen, V., Niemelä, E. and Taramaa, J. 1995. CACE + CASE = Better Reuse of Mechatronic Software. In: Muller, H.A. and Norman, R.J. (eds.), Proceedings of the Eight International Workshop on Computer-Aided Software Engineering (CASE'95), Toronto, Ontario, July 1995. IEEE Computer Society Press, Los Alamitos, California. Pp. 289 - 295.

Seppänen, V., Kähkönen, A-M., Oivo, M., Perunka, H., Isomursu, P. and Pulli, P. 1996. Strategic Needs and Future Trends of Embedded Software. TEKES Technology Review 48/96, Helsinki, Finland. 94 p.

Sherpa Corporation. 1995. Discipline in Software Development - White Paper, WPCC001 version 1, January 1995. 7 p.

Singh, A.K. and Irvine, M.D. 1995. CONSENS - An IT Solution for Concurrent Engineering. In: Sobolewski, A.J. (ed.), Proceedings of the Concurrent Engineering 95 (CE95), Johnstown, Pennsylvania, June 1995. Concurrent Technologies Corporation, Oakland, California. Pp. 635 - 644.

Soininen, J-P. 1995. VHDL and Software Maintenance. VTT Electronics, Oulu, Finland. Technical Report. 16 p. (in Finnish)

Sommerville, I. 1995. Software Engineering, Fifth Edition. Addison-Wesley Publishing Company, Wokingham, UK. 742 p.

Sommerville, I. and Dean, G. 1996. PCL: A Configuration Language for Modelling Evolving System Architecture. Software Engineering Journal, Vol. 11, No. 2, pp. 111 - 121.

Sperry, T. 1992. Checking out Version Control. Embedded Systems Programming, Vol. 5, No. 6, pp. 67 - 69.

Stankovic, J.A. 1996. Real-Time and Embedded Systems. ACM Computing Surveys, Vol. 28, No. 1, pp. 205 - 208.

Stefanelli, L. and Stroobants, L. 1991. Investigation of O&M activities, ESSDE Reference Facility Project ESTEC/ESSDE/STREAP/A11, Issue 1.1. 55 p.

Stragapede, A., Sarlo, L. and Coppola, P. 1997. Software & Systems Process Modelling in the European Industrial Space Standardization Context. In: T.-D. Guyenne (ed.), Proceedings of the 2nd International Conference on Data Systems in Aerospace (DASIA'97), Seville, Spain, May 1997. European Space Agency (ESA) SP-409, Noordwijk, the Netherlands. Pp. 449 - 454.

Swanson, E.B. 1976. The Dimensions of Maintenance. In: Proceedings of the 2nd International Conference on Software Engineering (ICSE2), San Francisco, California, October 1976. IEEE Computer Society Press, Washington, D.C. Pp. 492 - 497.

Swinehart, P.T., Zellweger, R., Beach, J. and Hagmann, R.B. 1986. A Structural View of the Cedar Programming Environment. ACM Transactions on Programming Languages and Systems, Vol. 8, No. 4, pp. 419 - 490.

Taramaa. J. and Purtilo J. 1990. Development of Embedded Software Using Interconnection Technology. University of Maryland, College Park, Maryland. Technical Report Series, UMIACS-TR-90-55, CS-TR-2455. 27 p.

Taramaa, J. and Seppänen, V. 1995. A Roadmap from Configuration to Application Management. In: Ross. M., Brebbia, C.A., Staples, G. and Stapleton, J. (eds.), Quality Management. Software Quality Management III Vol. 1, the 3rd International Conference on Software Quality Management, Seville, Spain, 3-5 April 1995. Computational Mechanics Publications, Southampton, UK. Pp. 111 - 120.

Taramaa, J. and Ketola, T. 1995. Process Modelling and Software Maintenance Aspects from the Point of View of Subcontractor in a Space Application. In: T.-D. Guyenne (ed.), Proceedings of an International Symposium on On-Board Real-Time Software (ISOBRTS'95), November 1995. European Space Agency (ESA) SP-375, Noordwijk, The Netherlands. Pp. 145 - 152.

Taramaa, J. 1997. Evaluation of an Extended Software Process Model for Space Applications. In: T.-D. Guyenne (ed.), Proceedings of the 2nd International Conference on Data Systems in Aerospace (DASIA'97), Seville, Spain, May 1997. European Space Agency (ESA) SP-409, Noordwijk, the Netherlands. Pp. 455 - 460.

Tervonen, I. 1994. Quality-Driven Assessment: A Pre-review Method for Object-Oriented Software Development. University of Oulu, Department of Information Processing Science, Research Papers, Series A 19, Oulu, Finland. Ph.D. Thesis. 133 p. + app. 7 p.

Thornton, J.D. 1996. Practical Description of Configurations for Distributed Systems Management. In: Proceedings of the 3rd International Conference on Configurable Distributed Systems (CDS96), Annapolis, Maryland, May 1996. IEEE Computer Society Press, Los Alamitos, California. Pp. 36 - 43.

Tichy, W. 1985. RCS - A System for Version Control. Software - Practice and Experience, Vol. 15, No. 7, pp. 637 - 654.

Tichy, W. 1988. Tools for Software Configuration Management. In: Winkler J.F.H. (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.

Tichy, W.F. 1994. Preface of Configuration Management. In: Tichy, W.F. (ed.) Configuration Management. John Wiley & Sons, Chichester, UK. 2 p.

Tiihonen, J., Soininen, T., Männistö, T. and Sulonen, R. 1995. State-of-the-Practice in Product Configuration - A Survey of 10 Cases in Finnish Industry. In: Tomiyama, M., Mäntylä, M. and Finger, S. (eds.), Proceedings of the First IFIP WG 5.2 Workshop on Knowledge Intensive CAD-1 (KIC-1), Espoo, Finland, September 1995. Pp. 147 - 164.

Tryggeseth, E., Gulla, B. and Conradi, R. 1995. Modelling Systems with Variability Using the PROTEUS Configuration Language. In: Estublier, J. (ed.), Lecture Notes 1005 in Computer Science, Software Configuration Management: ICSE SCM-4 and SCM-5 Workshops Selected Papers, International Workshop on Software Configuration Management (SCM5), Seattle, Washington, April 1995. Springer-Verlag, Heidelberg, Germany. Pp. 216 - 240.

Vierimaa, M. 1996. A Hypertext Approach to Application Understanding: A Maintenance Perspective. VTT Publications 274, Espoo, Finland: Technical Research Centre of Finland (VTT). Master's Thesis. 76 p.

Vidovic, N. and Ready, J. 1993. A Design Framework for Hardware-Software Co-Design of Embedded Systems. Real-Time Magazine 93/4, pp. 77 - 92.

Viskari, J. 1995. A Rationale for Automated Configuration Status Accounting. In: Estublier, J. (ed.), Software Configuration Management: ICSE SCM-4 and SCM-5 Workshops Selected Papers, Lecture Notes 1005 in Computer Science, International Workshop on Software Configuration Management (SCM5), Seattle, Washington, April 1995. Springer-Verlag, Heidelberg, Germany. Pp. 180 - 193.

Wein, M., MacKay, W.M., Gentleman, W.M., Stewart, D.A. and Gauthier, C.A. 1995. Evolution is Essential for Software Tool Development. In: Muller, H.A. and Norman, R.J. (eds.), Proceedings of the Eight International Workshop on Computer-Aided Software Engineering (CASE'95), Toronto, Ontario, July 1995. IEEE Computer Society Press, Los Alamitos, California. Pp. 196 - 205.

Weider, D., Yu, D., Smith, P. and Haung, S.T. 1990. Software Productivity Measurements. AT&T Technical Journal, Vol. 69, No. 3, pp. 110 - 120.

Westfechtel, B. and Conradi, R. 1998. Software Configuration Management and Engineering Data Management: Differences and Similarities. In: Magnusson, B. (Ed.), System Configuration Management, Lecture Notes 1439 in Computer Science, International Symposium on Software Configuration Management (SCM8), Brussels, Belgium, July 1998. Springer-Verlag, Heidelberg, Germany. Pp. 95 - 106.

Whitgift, D. 1991. Methods and Tools for Software Configuration Management. John Wiley & Sons, New York. 252 p.

Östlund, L. and Forssander, S. 1996. Management of a Flexible Software Production Based on Reusable Components. In: Proceedings of the Fifth European Conference on Software Quality, Dublin, Ireland, September 1996. Irish Quality Association and European Organization for Quality, Dublin, Ireland. Pp. 395 - 407.

Appendices of this publication are not included in the PDF version.
Please order the printed version to get the complete publication
(http://www.inf.vtt.fi/pdf/publications/1998)

Errata

Cover page	Is: Practical development of software configuration management for embedded system Should be: Practical development of software configuration management for embedded systems
Page 55, First chapter of 2.4.1	Is: Bennett et al. 19910 Should be: Bennett et al. 1991
Page 135, Reference	Is: Harjani, D.-R., Martelli, A. and Aquilino, D. 1995b. Lessons Learnt from Putting Process Modelling Techniques into Practice. In: Proceedings of Software. 5 p. Should be: Harjani, D.-R., Martelli, A. and Aquilino, D. 1995b. Lessons Learnt from Putting Process Modelling Techniques into Practice, In: Proceedings of Software Process Improvement Conference (SPI'95), Barcelona, Spain, December 1995, EU/ESSI Conference, 5 p.
Table 3 of Paper VI	Arrows are missing The following page shows the arrows of Table 3 of Paper VI

Table 3: The results of the evaluation of CM practices and their development.

	Configuration items (modules)	**Product data**	**Delivery Data**	**Customer Data**	**Error records**
No product management **0**	Electronics company-1	Electronics company-1	Electronics company-1		
Copying activity with manual versioning **1**	Machine/Process automation company-1	Machine/Process automation company-1			
Basic version management **2**		Machine/Process automation company-3	Machine/Process automation company-1		
Standardised version management **3**	*Electronics company-1* *Machine/Process automation company-1*		Telecommunication company-1 Machine/Process automation company-3 Telecommunication company-2		
Traceability over module, product and error data	Telecommunication company-1 Machine/Process automation company-3	*Electronics company-1*			
Basic configuration management **4**	*Telecommunication company-1*, 2 and 3 Machine/Process automation company-2	Telecommunication company-1, 2 and 3 Machine/Process automation company-2	*Telecommunication company-1* and 3 Machine/Process automation company-2		
Standardised configuration management **5**					
Basic product management **6**	*Telecommunication company-3*	*Telecommunication company-1*			
Total product management **7-11**		*Machine/Process automation company-3*	*Machine/Process automation company-3*		