

A roadmap from configuration to application management

J. Taramaa, V. Seppänen

VTT Electronics, PO Box 1100, FIN-90571, Oulu, Finland

Email: Jorma.Taramaa, Veikko.Seppanen@vtt.fi

Abstract

Software production problems have forced industrial organizations to assess the disciplines used to manage the software process in practice. Configuration management (CM) appears to be one of the software process activities that requires improvements. The importance of configuration management is clearly understood in industry, but there is yet no clear roadmap on how to proceed when improving CM practices. One of the biggest problems is that most organizations have considerable existing software assets to be maintained. Almost invariably this software has been produced using a large number of different methods, languages and tools, which makes its configuration management difficult.

Taking into account that new software has to be developed all the time, systematic procedures for evolving industrial software configuration management schemes are needed. We have developed an incremental approach to building CM environments, which has been evaluated and applied in cooperation with several industrial embedded systems manufacturers. In this paper, we describe the approach and the experiences in using it in various embedded computer system applications, such as mechatronic machines, and space instruments.



1 Introduction

Embedded computer programs are built-in control software for such high-value-added products as switching and production control systems, space instruments, communication devices, home electronics, and automated machines. Embedded software plays a central role in many modern products [1]. For example, in telecommunication systems it can constitute as much as 75 percent of the development costs [2]. The importance of software is also growing in mechatronic applications, where other technologies have traditionally taken a more central position [3]. A yet another important application area for embedded software is space instruments, as indicated for example by ESA's investments in the ESSDE project [4].

During the last few years the complexity of embedded software has remarkably changed. The size of software has also increased into hundred of thousands of lines of code. Table 1 shows as an example the evolution of the typical size of software in subsequent generations of mobile phones. Products based on embedded software are typically used for a long time and require several modernization cycles. A questionnaire concerning the maintenance of embedded software indicated that the current software generation will be used in several cases almost for the whole of the 90's [5]. The average age of embedded software in active use was about seven years, but some pieces of software had been used not less than for twenty years.

Table 1: Increase of software size in mobile phones.

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

Customers expect repairing and maintenance services from product manufacturers. For example, the mechanical parts, the hardware, the sensors and the actuators of an automated machine may have to be replaced several times. The software that controls such parts must then also be changed. In addition, some software-controlled product features may need optimization and tuning to accommodate to the new environment because a machine may also be resold and relocated several times.

During the last few years customer-oriented product development has become an important competing strategy. Companies developing embedded software can solve some of the customer-specific product tailoring problems using existing software development methods, tools, and environments. In the longer run, however, more effective approaches that provide better means for taking into account specific requirements set by customers are needed [6]. Indeed, software production problems have forced industrial organizations to assess the disciplines used to manage the software process in practice. Configuration management (CM) appears to be one of the software process activities that requires improvements.

The importance of configuration management is clearly understood in industry, but there is yet no clear roadmap on how to proceed when improving CM practices. One of the biggest problems is that most organizations have considerable existing software assets to be maintained. Almost invariably this software has been produced using a large number of different methods, languages and tools, which makes its configuration management difficult.

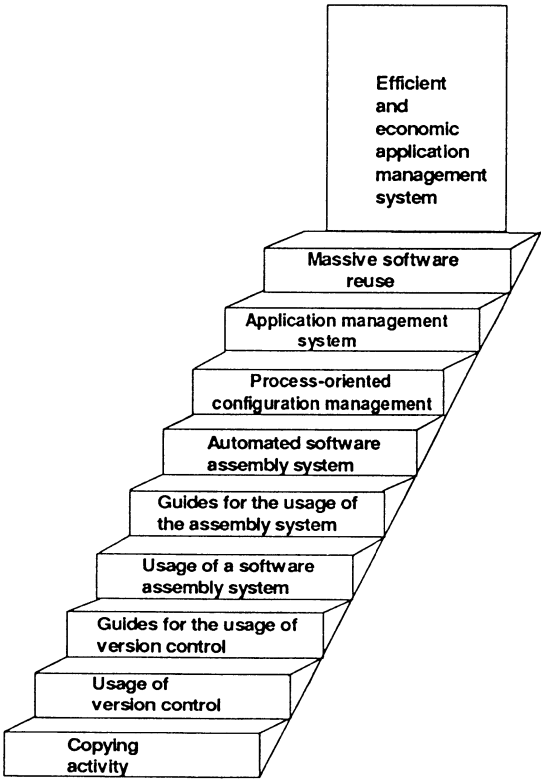


Figure 1: An incremental approach from simple SCM practices to application management.

Taking into account that new software has to be developed all the time, systematic procedures for evolving industrial software configuration management schemes are needed. We have developed an incremental approach to building CM environments, which has been evaluated and applied in cooperation with several industrial embedded systems manufacturers. In this paper, we describe the approach and the experiences in using it. One of the key aspects of the approach is a systematic roadmap from simple software configuration management practices to sophisticated application management methods, for the needs of software reuse and maintenance (Figure 1).

2 Configuration management

The traditional definition of software configuration management is given by Bersoff [7], according to which CM includes the following procedures:

- ☐ identifying and defining the items in the product (configuration identification),
- ☐ controlling the change of the items (configuration control),
- ☐ recording the status of the items (configuration status accounting), and
- ☐ verifying the completeness and correctness of the items (configuration audit).

In other words, software configuration management includes identifying, managing and controlling software as well as all software related descriptions, such as requirement and design specifications, test cases, test results, etc. Initially CM procedures were enforced using manual procedures, but the development of computer-aided tools during the eighties has made many routine configuration management tasks less error prone and much more efficient.

Although CM is essential throughout the entire product life cycle, its benefits are most obvious during maintenance of software products and components. Software maintenance involves changing the existing software and it is necessary to follow a systematic procedure for managing the changes. The main role of CM is to create and record linkages by which a maintainer can monitor and control proceeding from change requests to the implementation and testing of changes. These linkages include traceability, authorization, scheduling, and status accounting [8]. Therefore, CM systems used in software maintenance cannot be simple version control tools but need to provide for the management of evolving product families as part of a company's assets.

2.1 Version control-oriented configuration management

Version control provides a means to concurrently maintain specific instances of embedded software products developed and used in different environments. One of its practical benefits is saving of the archive space, because versions are stored

based on their incremental changes rather than full copies. Yet, the poor man's approach to CM based on saving full copies of different versions of software is still surprisingly common. The drawback of simple version control is that the maintainer still bears the full responsibility for managing all changes. Several commercial version control systems, often based on the traditional SCCS and RCS tools, are available.

Version control systems can be used in various individual ways. This will, in practice, introduce new problems which can only be managed by making operational guides for the usage of a specific version control system.

2.2 Assembly-oriented configuration management

A software assembly system consists of files, linkages between the files and compilation and linking commands stored in a configuration file. It compares the time stamps of object and source files and infers the commands that need to be done for assembling the software. An assembly system remarkably speeds up the process of producing software packages. First examples of this principle have been MAKE-based solutions, pioneered on UNIX systems and supported by the version control systems SCCS and RCS. A MAKE utility controls a build activity according to a script that defines dependencies between parts of the assembled software package. This has traditionally been named to the build-and-release activity.

An assembly system involves the same usage problem as a version control system, i.e. its everyday usage practices may become differentiated and become corrupted after a while. Therefore, creating common guides for the use of the assembly system are needed. Parts of such guides can be incorporated in the user interface of the system. In addition, the system can provide a help-like support mechanism including general and companywise CM instructions.

2.3 Automated assembly-oriented configuration management

The border between a guide-oriented assembly system and an automated software assembly system is in the implementation of the guides. A guide-oriented assembly system is manual. In an automated assembly system the guides have been automated as part of the mechanical assembly procedures.

A typical software assembly system is heavily based on software engineering techniques. In practice, it means that knowledge of the software engineering tools to be used in product deliveries are made explicit. Assembly automation provides for hiding such details, and thus facilitates reliability when repeating a product's assembly procedures, as well as ease of use by other than software professionals. The functions to be managed by assembly automation are, for example:

- ☐ generating and changing the assembled software,
- ☐ automated collection of the last versions for an assembly,
- ☐ usage of version control,

116 Software Quality Management

- ☐ tracking of customer orders and product data bases,
- ☐ manipulating product directories, and
- ☐ reporting various cross references for traceability.

2.4 Process-oriented configuration management

During the past few years software process development has gained more and more importance. Although practical software assembly systems have been developed, they do not necessarily consider the software processes used by particular product manufacturers. However, implicit models of software processes can be relatively easily linked to CM systems. Configuration management tools that include implicit process models have actually played quite an important role when advanced CM procedures have been integrated in state-of-the-art software engineering environments. There are already some commercial CM products available that include implicit process models, such as CaseWare, ClearCase, Aide-de-Camp and PCMS [9].

One of the basic problems in these environments is, however, their commitment to a fixed software process based almost entirely on the CM perspective. Flexibility demands the possibility to consider CM and software process requirements separately, although their relations have to be defined. Therefore, process-oriented configuration management should be seen as consisting of two sublevels, based on either fixed or flexible process models.

Moreover, embedded systems manufacturers have to take into account another product technologies in addition to software. Process aspects can thus be approached from the direction of the development of software or complete products. This issue will set new requirements for the second generation of commercial process-oriented CM tools.

3 From configuration to application management

3.1 Configuration management with efficient change management

Change management is a specific subject that is often conceptually separated from the other CM activities. However, in the most advanced current CM environments it is included as an inseparable part of comprehensive configuration management schemes.

Software maintenance requires change control solutions for specific maintenance functions, such as problem understanding, localization, solution analysis, and impact analysis. These maintenance functions involve specific application management problems, in addition to general configuration management concerns. Therefore, application management is a wider concept than the traditional definition configuration management as given above. The definition of application management proposed by the Esprit3 project AMES is as follows [10]:

"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".

Relations between software maintenance and configuration management are illustrated in Figure 2. In addition to the software archive and other basic CM features, a strong link between maintenance and CM is change control on the basis of which several maintenance activities are actually implemented. Application management includes as one of its most important aspects the definition of the process to be used in software maintenance.

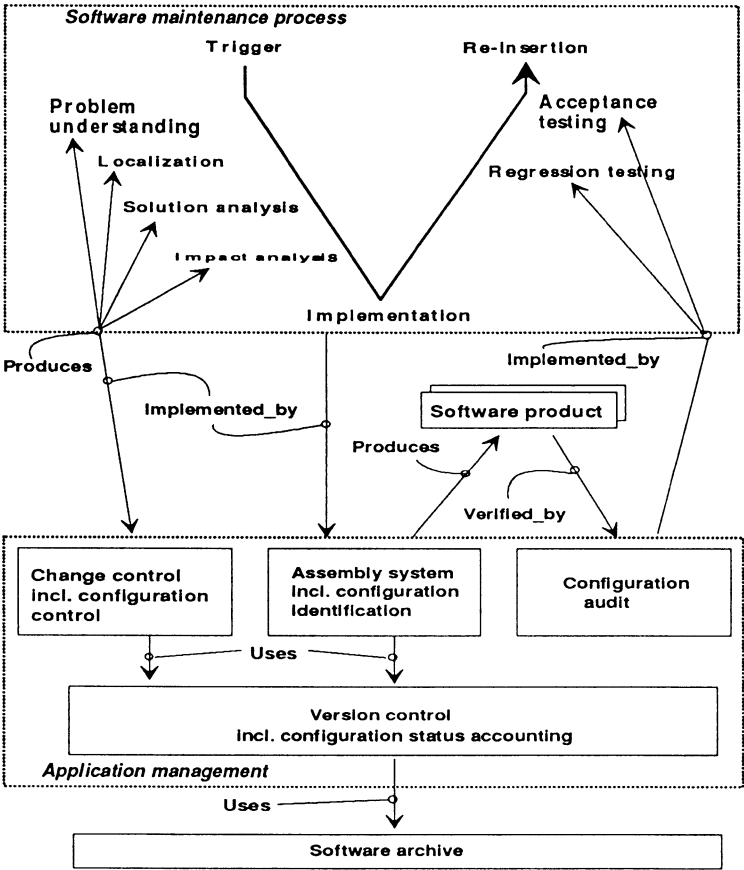


Figure 2: Relations between the software maintenance process and application management.

3.2 Reuse-oriented application management

Systematic reuse is often considered as one of the most important enabling technologies for improving software quality and productivity. From the viewpoint of configuration management, software reuse results in yet another requirements. In addition to the software assembly functions, the CM system has to support the search and selection of the potentially reusable software modules. One of the key concepts for effective software reuse is to understand the domain of applications involved [11]. The basic idea of massive reuse is to develop various product properties with explicit reusability in mind. A project dealing with mechatronics systems provided us for plenty of practical experiences in the role of CM in reuse [3]. As an example, a complete set of guidelines for the identification, documentation and management of reusable mechatronic software components were developed.

4 Experiences in the application of the approach

Experiences in the incremental approach from CM to application management have been gained from three different embedded systems development projects. The first of the projects, discussed in [3], provides for an example where software is only one of the product technologies to be applied. In addition, software was not the core technology of the product and was thus actually implemented by a subcontractor. However, maintenance responsibility for the software was allocated solely to the product manufacturer. In this case study, the development of CM practices ranged from taking a commercial version control in use to the first experiments of an automated assembly system.

The second project was in its planning phase, when we were hired to assess the level where the participating companies were in CM [12]. These companies were already using CM procedures, but general developments in computer-aided software engineering environments and greater expectations concerning CM had negatively changed their position. Typically, these companies were dealing with professional electronics goods and telecommunication applications. The necessary CM development activities in these companies involve automated assembly systems and software reusability.

The third project, AMES [10], has brought well-defined software maintenance models to the context of CM, where the traditional CM functions consist part of application management. The extensions that are needed, in practice, include management of changes during the analysis of maintenance requirements [13], definition of the maintenance process with regard to the implementation and testing of changes, and tool definitions for CM in maintenance.

5 Conclusions

This paper discussed an incremental approach to configuration management, illustrated as ladders from simple CM procedures to advanced application

management practices. When maintaining embedded software, software modules developed using various languages, methods and tools must usually be managed. Typically, the modules to be maintained have been developed during the period of several years, even tens of years. Maintenance resources are in the most cases limited. All these things favor an incremental approach to developing CM as part of improvement of the maintenance process. A single step in such an approach results in a visible baseline where a company can assess the need of further work and more advanced tools. The risks of failing in the development of change and configuration management practices can be effectively limited.

In the incremental approach it is essential to understand the state-of-the-practice in the company concerning CM, the domain of application of the software to be maintained, the role of software maintainers in the context of all product technologies, because all these constrain the CM tools that can and should be obtained. CM investments are typically long-term decisions and it is difficult and costly to withdraw them.

ACKNOWLEDGEMENTS

The work presented in this paper has been carried out in several projects during last two years. One of the most important of those projects is AMES (Application Management and Support and Environment Systems). In addition, work in the national projects LOKKI (Assembly of Machine Control Software) and LEIVO (Management of Customized Versions of Embedded Software) have provided insights for writing this paper. AMES is partially funded by the European Communities under the ESPRIT program, project no. 8156, Technical Development Centre of Finland (TEKES) and VTT Electronics. The AMES consortium is formed by Cap Gemini Innovation (France), Cap Programator (Sweden), University of Durham (UK), Intecs Sistema Spa (Italy), Matra Marconi Space (France), OPL-TT (Spain), Space Systems Finland (Finland), and VTT Electronics (Finland). LOKKI was mainly funded by TEKES and VTT Electronics, whereas LEIVO is funded by TEKES and a number of Finnish electronics firms.

References

- [1] Davis, A.M. *Software Requirements - Analysis & Specification*, Prentice-Hall, Englewood Cliffs, New Jersey, 1990.
- [2] Weider, D., Yu, D., Smith, P., and Haung, S.T., Software Productivity Measurements, *AT&T Technical Journal*, May/June 1990, pp. 110-120.
- [3] Taramaa, J., Lintulampi, R., Seppänen, V., Automated Assembly of Machine Control Software, *Mechatronics*, Vol. 4, No. 7, 1994, pp. 753-769.
- [4] Favaro, J., Coene, Y., Casucci, M., The European Space Software Development Environment Reference Facility Project: A Status Report, ACM SIGSOFT, *Software Engineering Notes*, Vol. 19, No. 2, 1994, pp. 68-71.

120 Software Quality Management

- [5] Taramaa, J., Oivo, M. Evaluation of Software Maintenance of Embedded Computer Systems, *International Symposium on Engineered Software Systems*, Malvern, Pennsylvania, May 1993, pp. 193-203.
- [6] Rosenthal, S.R., *Effective Product Design and Development*, Business One Irwin, Homewood, Illinois, 1992.
- [7] Bersoff, E.H., Henderson, V.D., Siegel, S.G. *Principles of Software Configuration Management*, Prentice-Hall, Englewood Cliffs, New Jersey, 1979.
- [8] Zvegintsov, N. Software Configuration Management: Control for the Software Team, *Software Management News*, Vol. 11, No. 3, 1993, pp. 13-24.
- [9] Ingram, P., Burrows, C., Wesley, I., Configuration Management Tools: a Detailed Evaluation, *OVUM report*, 1993.
- [10] AMES Application Management Environments and Support, Esprit 3 Project 8156, Technical Annex, Version 1.0, August 1993.
- [11] Frakes, W.B., Isoda, S., Success Factors of Systematic Reuse, *IEEE Software*, Vol. 11, No. 5., 1994, pp. 70-79.
- [12] Taramaa, J., Seppänen, V., Auer, A., LEIVO - Management of Customized Versions of Embedded Software, Project Proposal, VTT Electronics, 1994 (in Finnish).
- [13] Mäkräinen, M., Taramaa, J., Quality Aspects in Evolution of Real-time Embedded Software, submitted to SQM95 for publication, 9 p.