Evaluating the Software Configuration Management Process

Antti Kokkonen

University of Tampere
Department of Computer and Information Sciences
Master's thesis
June 2003

University of Tampere
Department of Computer and Information Sciences
Author: Antti Kokkonen
Master's thesis, 52 pages.
June 2003

Abstract

The purpose of the thesis is to identify Software Configuration Management (SCM) definitions, concepts and models based on previous studies, and to familiarize one with the SCM process in general. This knowledge is used to analyze, evaluate and improve the SCM towards continuous improvement of the software process. Research questions of the thesis are: "What is SCM?" and "How can the software configuration management process be evaluated and continuously improved?". To answer these questions, the study applies conceptual-analytical research approach: The recommendations and requirements for SCM are gathered from various standards and other publications. As a result of the study, this thesis presents guidelines for defining goals for SCM, evaluating the SCM process and creating the SCM evaluation plan. The guidelines can be used when adopting new SCM technology, and when evaluating an existing solution. The guidelines give an introduction to the process and ideas on how to adapt SCM in research and work.

Keywords: software configuration management (SCM), software process evaluation, software process analysis, and software process improvement (SPI).

Tampereen Yliopisto
Tietojenkäsittelytieteiden laitos
Kirjoittaja: Antti Kokkonen
Pro Gradu -tutkielma, 52 sivua.
Kesäkuu 2003

Suomenkielinen tiivistelmä (Finnish abstract)

Tutkielman tarkoitus on antaa yleiskatsaus ohjelmiston konfiguraationhallintaan esittelemällä aikaisempien tutkimusten perusteella konfiguraationhallinnan määritelmät, pääkohdat ja mallit. Tätä tietoa voidaan käyttää konfiguraationhallinnan analysointiin, arvioimiseen ja kehittämiseen kohti jatkuvaa prosessin kehitystä. Tutkimuskysymyksinä on: ”Mitä on ohjelmiston konfiguraationhallinta?” ja ”Kuinka konfiguraationhallintaprosessia voidaan arvioida ja kehittää?”. Vastatakseen näihin kysymyksiin, tutkielmassa sovelletaan teoreettis-käsitteellistä tutkimusmetodia: Konfiguraationhallintaa koskevia suosituksia ja vaatimuksia on kerätty eri ohjelmistotuotannon standardeista ja muista julkaisuista. Tutkimuksen tuloksena tutkielmassa esitellään ohjeita konfiguraationhallinnan tavoitteiden määrittämiseen, prosessin arvointiin ja konfiguraationhallinnan arviointisuunnitelman tekemiseen. Ohjeita voidaan soveltaa sekä uuden konfiguraationhallintateknologian käyttöönotossa että olemassa olevan sovelluksen arvioinnissa. Tutkimus antaa johdannon ohjelmiston konfiguraationhallintaprosessiin ja ideoita, kuinka sitä voi käyttää teollisuudessa ja akateemisessa tutkimuksessa.

Avainsanat: Ohjelmiston konfiguraationhallinta, ohjelmistotuotantoprosessin arvointi, prosessin analysointi ja prosessin kehittäminen.

Contents

1. Introduction 1
   1.1. What is SCM? 1
   1.2. Research Goals 2
   1.3. Methods 4
   1.4. Previous studies 4
   1.5. Outline of the study 4
2. Software Configuration Management 5
   2.1. Concepts in Software Configuration Management 5
   2.2. Activities in Software Configuration Management 12
   2.3. Standards for Software Configuration Management 15
   2.4. Software Configuration Management Models 20
3. Goals for SCM 24
   3.1. Using standards in goal definition 25
   3.2. Goals for the organization 28
   3.3. Requirements for the SCM tools 28
   3.4. Goals for agile SCM 30
   3.5. SCM goals summarized 31
4. Evaluating the SCM Process 32
   4.1. Evaluating the organization 34
   4.2. Evaluating the tools for SCM 35
   4.3. Guidelines for Evaluation 36
5. SCM Evaluation Plan 37
   5.1. Conducting the evaluation for SCM 38
   5.2. SCM Evaluation and Improvement plan 39
   5.3. Evaluation Plan Guidelines 42
6. Discussion 43
   6.1. Recommendations and restrictions 45
   6.2. Future research 45
References 46
Web resources 48

1. Introduction

Software engineers build computer software. Software consists of programs, documents, and data. To build a successful product that meets the requirements of the people who will be using the program, software engineering approach with planning and quality assurance is usually applied. During the life cycle of the software project, the software and its components are likely to change. To ensure that the changes do not turn the process into chaos, the change must be efficiently controlled. The changes are controlled by a software engineering activity called Software Configuration Management (SCM).

Everyone involved in the software engineering process is at some point involved with the changes of the components, and therefore in touch with SCM. As SCM begins when the project begins and ends only when the software is taken out of operation, it is an essential part of good project management and solid software engineering practice. [Pressman, 1997]

The adoption, evaluation and improvement of SCM technology are key components of process maturity improvement. SCM provides a solid base to any software organization or product to evolve effectively and under control. Because of this, SCM can be seen as one of the most important practices in software engineering.

1.1. What is SCM?

The purpose of Software Configuration Management is to establish and maintain the integrity of the products of the software project throughout the project's software life cycle.

Software Configuration Management (SCM) provides a mechanism for identifying, controlling and tracking the versions of each software item. In many cases earlier versions still in use must also be maintained and controlled. [ISO9000-3]

A standard definition of configuration management [IEEE, 1990] mandates the following SCM procedures [Dart, 1991]:

Identification. Reflects the structure of the product, identifies components and their type, making them unique and accessible in some form.

Control. Controls the release of a product and changes to it throughout its life cycle by having controls in place that ensure reliable software via the creation of a baseline product.

Status Accounting. Records and reports the status of components and change requests, and gathers vital statistics about components in the product.

Audit and Review. Validates the completeness of a product and maintains consistency among the components, ensuring that the product is a well-defined collection of components.

Dart [Dart, 1991] expanded the standard definition to include procedures like construction management, process management, and team work control:

Manufacture. Manages the construction and building of the product in an optimal manner.

Process Management. Ensures the carrying out of the organization's procedures, policies and life cycle model.

Team work. Controls the work and interactions between multiple developers.

1.2. Research Goals

The purpose of this thesis is to identify SCM definitions, concepts and models based on previous studies, and to familiarize one with the SCM process in general. The other purpose is to encourage the organizations to constantly evaluate the methods and procedures used in the software development, and this way to improve the overall process.

One part of my current job is the management of the software building, which offers me a higher-level view to the SCM process. Because of this, the thesis reflects the upper, and more abstract level of the SCM process. I believe that the software development processes, like SCM, should always be critically analyzed and improved.

With these aspects in mind, I present what should be taken into account when being involved with SCM. I keep the focus of the study on SCM, but as SCM is somehow involved in almost every aspect in the software development; some aspects of SCM inevitably reflect to other software engineering activities as well. I also see SCM as the first step towards more mature software development. This way the actions and guidelines work towards continuous improvement of SCM. The problem domain is illustrated in Figure 1.2-I.

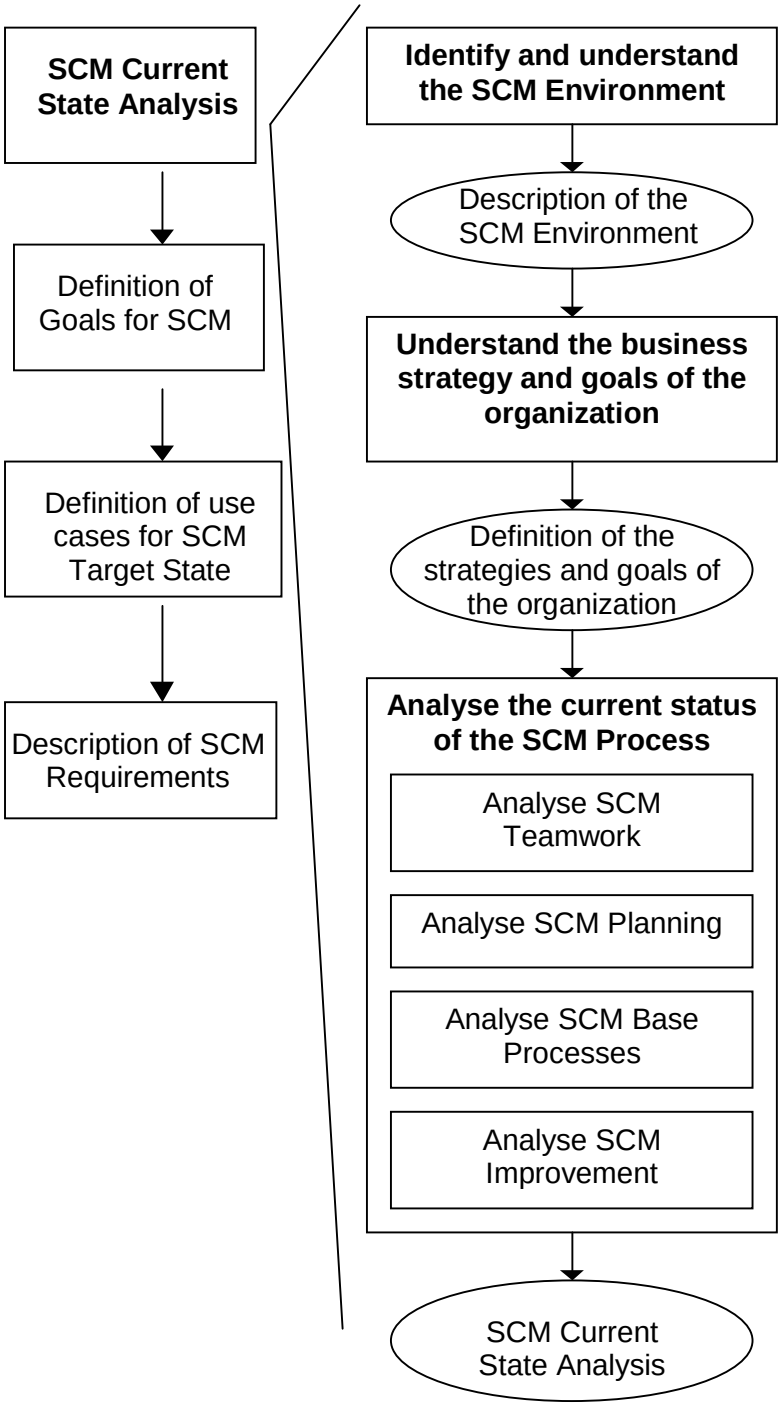

Figure 1.2-I. Problem Domain. Modified from [Rahikkala, 2000].

The research questions to be answered in this thesis are:

- What is SCM?
- How can the software configuration management process be evaluated and continuously improved?

1.3. Methods

The purpose of the study is to define what SCM is, in other words we answer the question “How things are done or what SCM is like”. To some extent the study will also look “How things should be done or what SCM should be”. To answer these questions, the study applies conceptual-analytical research approach: The recommendations and requirements for SCM are gathered from various standards and other publications. [Järvinen, 2000]

1.4. Previous studies

Several authors have published articles and books that offer basic knowledge and information about SCM ([Berlack, 1992], [Pressman, 1997]). Software configuration management has also been defined by several standards ([IEEE, 1990], [ISO9000-3]). The activities and user roles of SCM have been recognized [Dart, 1991], and several SCM models have been formed [Feiler, 1991].

The improvement and evaluation of SCM is an important part of Software Process Improvement. SCM has been adapted towards distributed organizations [Rahikkala, 2000] and also into development of embedded software [Taramaa, 1998]. Several authors have published studies of evaluating and improving SCM process [Mosley, 1995][Dart, 1996], choosing SCM tools [Berlack, 1995] [Mosley et al, 1996] and adopting SCM technology [Kinsbury, 1996].

1.5. Outline of the study

SCM is recognized as an important part of software engineering and has been well covered in the literature; this thesis reviews those solutions, definitions and models. Previous studies offer various methods to evaluate and improve the SCM process. These different practices and standards are combined in a SCM evaluation and improvement plan.

In Chapter 2 the concepts of SCM are introduced (Activities recognized in SCM, definition of the SCM process, standards for SCM, and SCM models). Chapter 3 presents the goals for SCM as presented in standards. Chapter 4 shows how the process is evaluated against the defined goals. Chapter 5 combines the standards and related studies with the previous chapters to form a SCM evaluation and improvement plan. The thesis is summarized and the results are discussed in Section 6.

2. Software Configuration Management

"There is nothing permanent except change."

- Heraclitus 500 B.C.

Change is inevitable in a software project, but how do we make sure that the changes are done properly? Doing changes properly could mean that the changes are evaluated before they are made, recorded before implementation, reported to those with a need to know and controlled in a manner that reduces errors and therefore improves quality. Several changes could happen at the same time, by several software engineers, and from different countries. Solution to this probable chaos could come from Software Configuration Management (SCM).

2.1. Concepts in Software Configuration Management

SCM is a set of activities that have been developed to manage change throughout the life cycle of computer software. [Pressman, 1997]

To clarify the SCM process and to understand the software configuration management and its activities in general, we must identify some concepts commonly used when describing the SCM environment.

The terminology of SCM in literature is not consistent. Terms can have different definitions or multiple meanings depending on the writer. In this section the principal SCM terms are presented according to IEEE standards 610.12-1990 Standard Glossary of Software Engineering Terminology and 828-1990 Standard for Software Configuration Management Plans [IEEE, 1990].

- **Configuration** is the arrangement of a computer system or component as defined by the nature, number and interconnections of its constituent parts.
- **Component** means one of the parts that make up the system. Components may be further subdivided to other components resulting in a hierarchical representation of the sys-tem. The definition of configuration means that in order to specify a configuration we must know all the parts, identified, e.g., by their name and version number, and relationships between the parts belonging to the desired configuration.
- **Configuration management** is a discipline applying technical and administrative direction and surveillance to: identify and document the

functional and physical characteristics of a configuration item, control changes to those characteristics, record and report change processing and implementation status, and verify compliance with specified requirements.

- **Configuration item** (CI) is an entity treated separately in the configuration management process.

Terms **version**, **revision** and **variant** are commonly used in SCM literature. Version is a common term meaning either a revision or a variant. Revision is a version that replaces a previous version. Variant is a version that can be used as an alternative to another version.

2.1.1. Software Configuration Item (SCI)

A Software Configuration Item (SCI) is a collection of software elements, treated as a unit, for the purpose of configuration management. Several factors may be relevant in deciding where to draw the boundaries of a software configuration item. SCI may be any kind of software item, for example: source code, 3D-model, graphics, a module, a document, or a set of SCIs.

Different variants of SCIs should be stored and placed under version control (Subsection 2.2.6.). The version control also enables the different configurations of the software. Also the naming scheme established for the SCIs should incorporate the version number. [Pressman, 1997]

2.1.2. Baselines

A baseline is a software configuration concept that helps to control change without seriously impeding justifiable change. [Pressman, 1997]

The IEEE standard defines a baseline as:
A specification or product that has been formally reviewed and agreed upon, that thereafter serves as the basis for further development, and can be changed only through formal change control procedures. [IEEE, 1990]

Baseline is a document or a product (in other words, a Software Configuration Item). Baselines provide a stable basis for continuing evolution of configuration items. Baselines are added to the configuration management system as they are developed. Changes to baselines and the release work products built from the configuration management system are systematically controlled and monitored via the configuration control, change management, and configuration auditing functions of configuration management. [CMM, 1993]

2.1.3. Definitions for SCM

Software configuration management is both a managerial and a technical activity, and it is essential for proper software quality control. The software configuration management activities for a project are defined in the Software Configuration Management Plan (SCMP).

According to the traditional definition used by IEEE 828-1990 [IEEE, 1990], SCM includes the following basic elements (also illustrated in the Figure 2.1-I below):

Configuration Identification. Reflects the structure of the product, identifies components and their types, making them unique and accessible in some form.

Change Control including Configuration Control. Controls the release of a product and changes to it throughout the software life cycle and ensures consistent software via the creation of a baseline product.

Controlling Status Accounting. Records and reports the status of components and change requests, and gathers vital statistics about components in the product.

Configuration Audit and Review. Validates the completeness of a product and maintains consistency among the components, ensuring that the product is a well-defined collection of components.

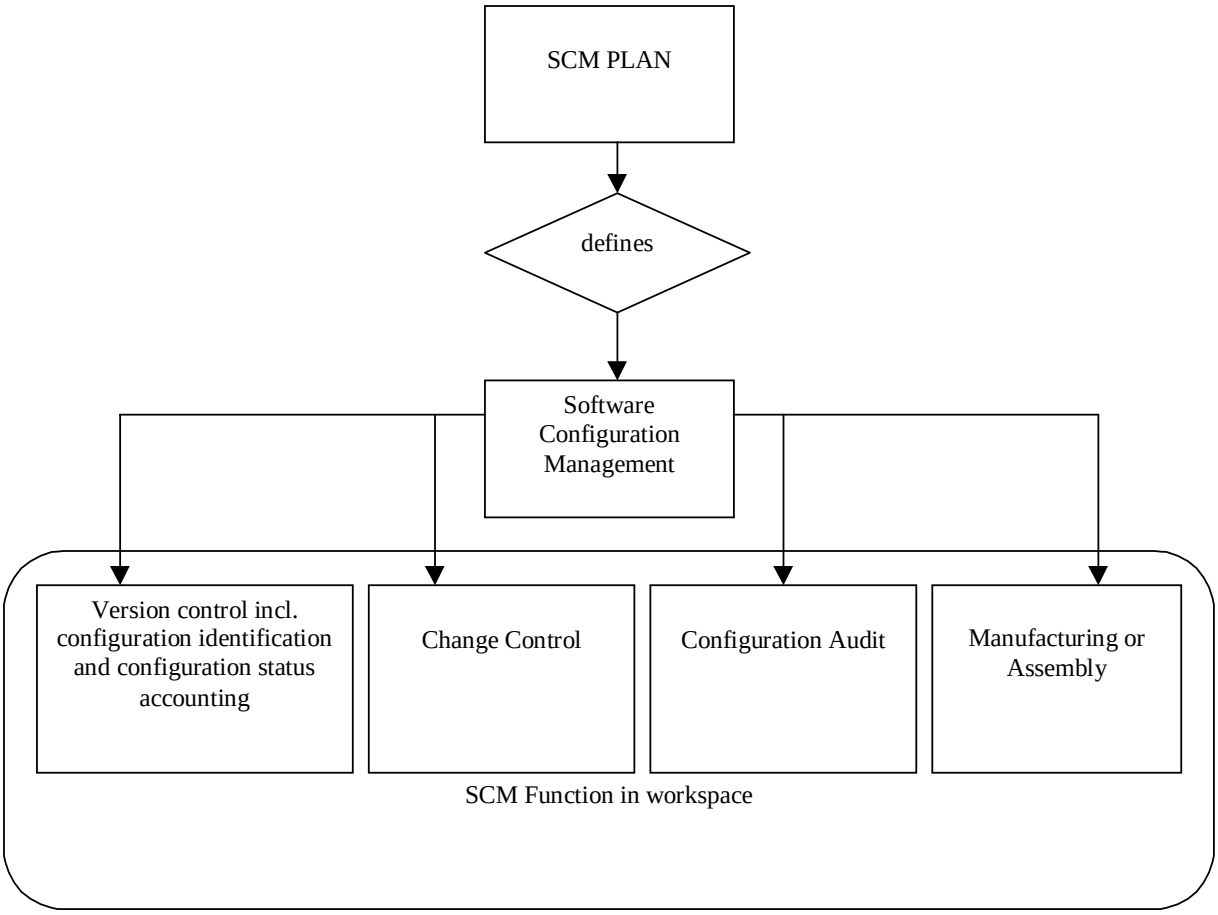

Figure 2.1-I. The traditional definition of the SCM Process. Modified from [Rahikkala, 2000].

This traditional definition has been extended by Dart [Dart, 1991] to include manufacturing issues (generating derived configurations by a build mechanism), process management and team work control:

- **Manufacture.** Manages the construction and building of the product in an optimal manner.
- **Process Management.** Ensures that the organization's procedures, policies and life cycle model are followed.
- **Team work.** Controls the work and interactions between multiple developers.

After these extensions SCM can be seen as an activity, which controls change through the functions of component identification, change tracking, version selection and baselining, software manufacture, and management of simultaneous updates. The definitions are summarized in the Figure 2.1-II [Taramaa, 1998].

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

Figure 2.1-II. The SCM definition categories, SCM elements and descriptions summarized [Taramaa, 1998].

When relating Feiler's classification to these definitions [Feiler, 1991], the traditional definition sees SCM more as a management discipline and the extended definition sees SCM as a development discipline. [Taramaa, 1998]

The various definitions for the SCM can also be described as different models. The models usually cover the whole SCM environment, but the different

models focus on different aspects. As companies or software projects can have different strategies or goals, the SCM models can offer a simplified way to decide how one wants to look at the software project and process. Some different models are presented in Section 2.4.

2.1.4. Software Process Improvement

The SCM process should also be reviewed, analyzed and improved. To improve the process, we must define the requirements we want the process to meet. There are many possibilities how to do this and how to check how the requirements were met. To give an example, the Goal / Question / Metric approach is used to specify the needs (goals) for the improvement and make sure the requirements (questions) are met. [Basili et al, 1994].

Successful changes to the software process start at the top of the organization. Senior management leadership is required to launch a change effort and to provide continuing resources and impetus, although ultimately, everyone in the organization is involved. Software Process Improvement and Capability determination summaries software improvement basics as follows [SPICE]:

- Software process improvement demands investment, planning, dedicated people, management time and capital investment.
- Process improvement is a team effort – those not participating may miss the benefits and may even inhibit progress.
- Effective change requires an understanding of the current process and a goal – you must know where you are and where you want to be.
- Change is continuous, not a one-shot effort – it involves continual learning and evolution.
- Software process changes will not be sustained without conscious effort and periodic reinforcement.

2.1.5. The functionality areas in SCM

There are different kinds of users of SCM systems. A user is given a specific role and can have a different view of SCM and, hence, different requirements for a SCM system. Dart [Dart, 1991] has recognized the following functionality areas / roles (Illustrated in Figure 2.1-III):

(Team-Centered areas ~ Technical aspects)

- **Components:** Identifies, classifies, stores and accesses the components that make up the product.
- **Structure:** Represents the architecture of the product.
- **Construction:** Supports the construction of the product and its artifact.
- **Team:** Enables a project team to develop and maintain a family of products.

(Process-Centered areas ~ Management issues)

- **Auditing:** Keeps an audit trail of the product and its process.
- **Accounting:** Gathers statistics about the product and the process.
- **Controlling:** Controls how and when changes are made.
- **Process:** Supports the management of how the product evolves.

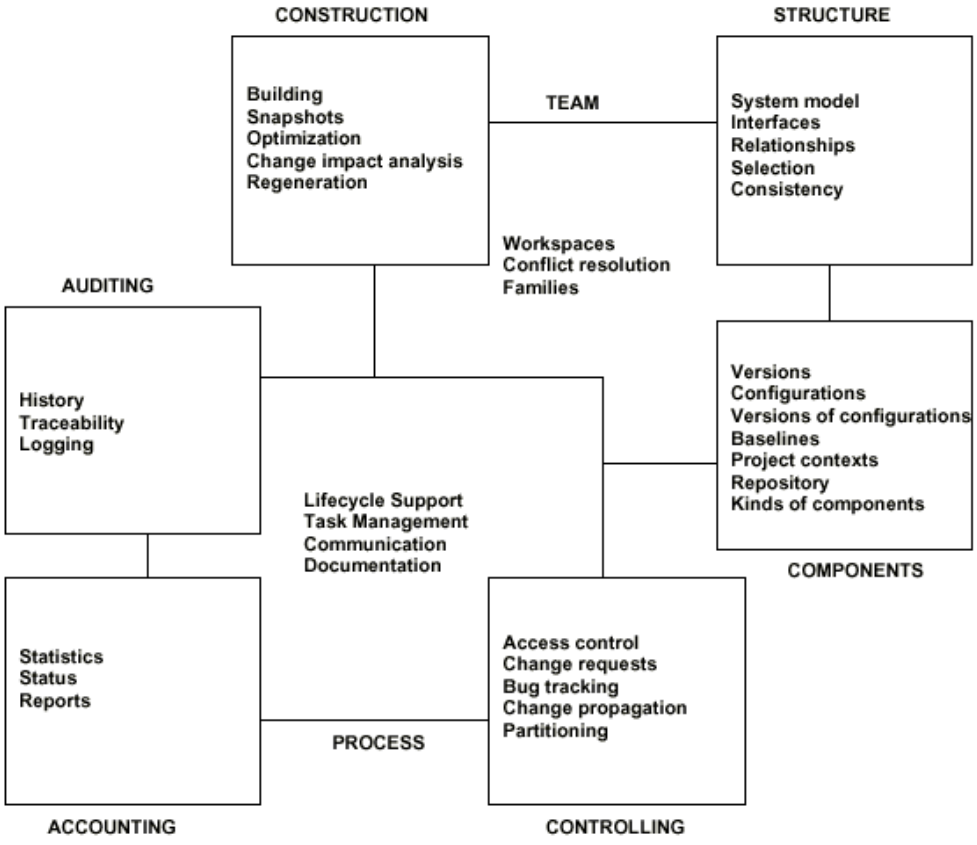

Figure 2.1-III. The SCM functionality requirements [Dart, 1991]

2.2. Activities in Software Configuration Management

A Software Configuration Management Plan defines the project strategy for SCM (Basic elements are illustrated in Figure 2.2-I). The SCM Plan describes all the activities, responsibilities, procedures and tools used in management of the product development. The SCM Plan identifies the SCM requirements, version control procedures, controlled items, and change control used with those items.

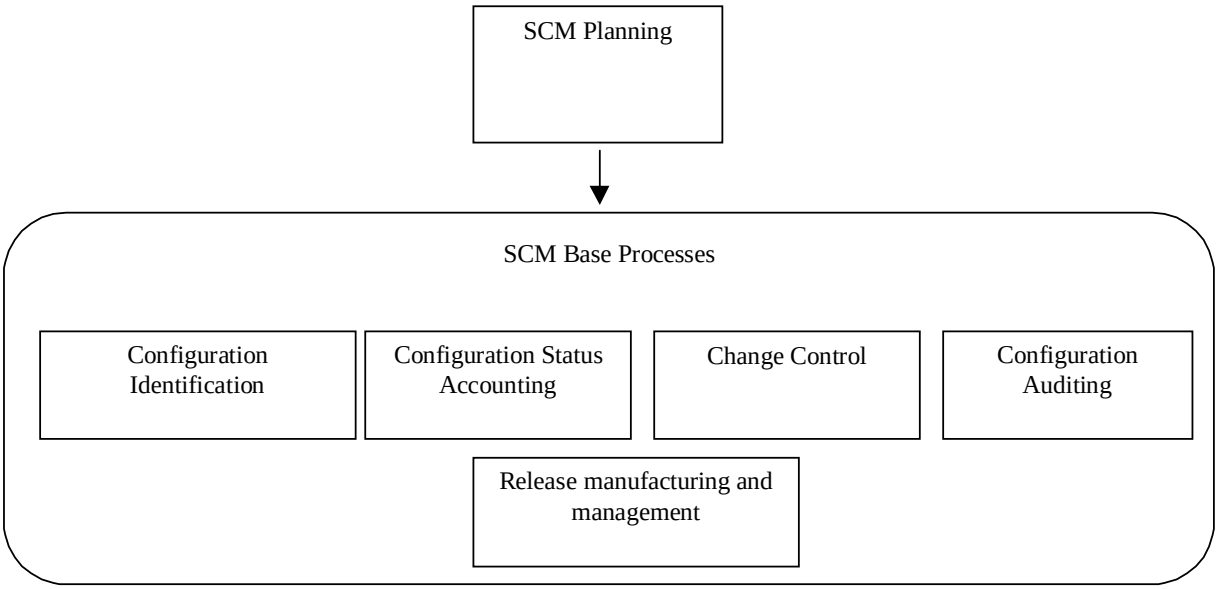

Figure 2.2-I. The basic elements of software configuration management [Taramaa, 1998].

2.2.1. Configuration Identification

The purpose of configuration identification is to identify the versions of each software item, which together constitute a specific version of a complete product [ISO9000-3].

Examples of work products that may be placed under configuration management include the following:

- Plans
- Process descriptions
- Requirements
- Design data
- Drawings
- Product specifications
- Code

- Compilers
- Product data files
- Product technical publications.

2.2.2. Configuration Status Accounting

The purpose of the Configuration Status Accounting activity is to identify the build status of software products [ISO9000-3].
Software configuration status accounting is the administrative tracking and reporting of all configuration items. The status of all configuration items shall be recorded. Configuration status accounting continues, as do all other configuration management activities, throughout the software life cycle.

To perform software status accounting, each software project shall record the:

- Date and version / issue of each baseline
- Date and status of each Review Item Discrepancy (RID) and Document Change Record (DCR)
- Date and status of each Software Problem Report (SPR), Software Change Request (SCR) and Software Modification Report (SMR)
- Summary description of each Configuration Item (CI).

Configuration status accounts should be produced at project milestones, if such have been established (as recommended by several standards), and may be produced periodically between project milestones.

2.2.3. Change Control

The purpose of the change control activity is to:

- Control simultaneous updating of given software item by more than one person.
- Provide coordination for the updating of multiple products in one or more locations as required.
- Identify and track all actions and changes resulting from a change request, from initiation to release.

[ISO9000-3]

Software change control (including configuration control) is the process of evaluating proposed changes to configuration items (SCI) and coordinating the implementation of approved changes. Software configuration control of an item can only occur after formal establishment of its configuration identification and inclusion in a baseline.

Proper software configuration control demands the definition of:

- The level of authority required to change each SCI
- The methods for handling proposals for changing any SCI.

2.2.4. Configuration Auditing

Like the software configuration management as a whole process, especially configuration auditing is an important activity when considering the quality assurance of the software product. Therefore in some cases where SCM is a formal activity, a separate quality assurance group can also conduct configuration audit.

The configuration auditing is usually supplemented by formal technical reviews. The formal technical review focuses on the technical correctness of the configuration object (or item) that has been modified. The software configuration audit complements the formal technical review by assessing a configuration object for characteristics that are generally not considered during review. [Pressman, 1997]

2.2.5. Release manufacturing and management

Release methods and tools are documented in the SCM Plan. When managing the releases, the procedure stated in the plan is to be followed. Manufacturing the release using different variants of the same SCI is where configuration management finally produces something out of the process. Usually the release product is sent to testing or customer deliveries. The product life cycle does not stop there. The tested product can be corrected, if errors were found in the testing. The product can be developed further, configured with other attributes (or version variants), and so on.

2.2.6. Version Control

Clemm [Clemm, 1989] described the version control in the SCM as follows:
Configuration management allows a user to specify alternative configurations of software systems through the selection of appropriate versions. Associating attributes with each software version supports this and describing the set of desired attributes allows a configuration to be specified (and constructed).

The attributes mentioned above can be as simple as a specific version number that is attached to each object. The versioning system used in a project or a

company should be carefully established and documented (for example in the software configuration management plan).

Version control is an essential part for SCM and therefore for the whole software project. Versions of SCIs are understood to be either variants or revisions. With versioned SCIs, we have a way to examine and control the change through the revisions. Version control also enables the configuration of the software in a form of variants.

The software project can take the benefit from the use of several version variants. For example, take a product that is delivered to various countries exactly the same, except for the user interface (for language reasons). We can compile the “base” product just once, because it is similar in every release. After this we can finalize the release for each country by adding a different variant of the user interface component to the product.

2.3. Standards for Software Configuration Management

The three standards described in this section, especially the Capability Maturity Model (CMM), aim at guiding software organizations in selecting process improvement strategies. An organization should first determine the current process maturity before identifying critical quality and process improvement issues.

The standards, as well as a great deal of SCM literature are written mainly for large projects and can impose a lot of bureaucracy on development process if implemented too literally. Applying the standards in companies is not straightforward, but requires a great amount of balancing between the amount of control and flexibility.

2.3.1. IEEE 828

IEEE standard 828 defines the minimum content of Software Configuration Management Plans (SCMP). Like most standards, it gives guidelines to be followed, not specific instructions. IEEE 828 does give guidelines what to put into the SCMP, but it doesn't tell how to do specific SCM activities. [IEEE, 1990]

According to IEEE 828 the SCM plan should identify reviews, responsibilities, and integration of software. In addition to this, SCM plan should describe methods used for:

- Identification of software configuration items,
- Control and implementation of change,

- Recording and reporting change and problem report implementation status,
- Conducting configuration audits,
- Review and approval cycles as well as approval authority, and
- Identification of personnel responsible for configuration management.

IEEE standard 1047 ("Sub-standard" of IEEE 828) is described as a guide, which provides guidance in planning software configuration management practices that are compatible with IEEE 828. Standard 1047 is meant for the developers of the software, for the management, and for those responsible for creating the SCM Plans. Like some have noticed, standard also states that SCM can support a software engineering process in many different ways, and the SCM Plan can be tailored to the needs and resources of any project. [IEEE, 1987]

2.3.2. ISO 9000-3

The standard ISO 9000-3 is a quality assurance standard, which gives guidelines to the development, supply and maintenance of software. According to this standard SCM should [ISO9000-3]:

a) identify uniquely the versions of each software item,
b) identify the versions of each software item, which together constitute a specific version of a complete product,
c) identify the build status of software products in the development and products that have been delivered and installed,
d) control simultaneous updating of a given software item by more than one person,
e) provide coordination for the updating of multiple products in one or more locations, and
f) identify and track all actions and changes resulting from a change request, from initiation to release.

2.3.3. Capability Maturity Model

Based on several years of experience with software process improvement, the Software Engineering Institute (SEI) at Carnegie Mellon University, Pennsylvania, published the first Capability Maturity Model (CMM) in 1991. [CMM, 1993]

CMM describes the Configuration Management process to involve the following [Paulk et al, 1993]:

- Identifying the configuration of selected work products that compose the baselines at given points in time.
- Controlling changes to configuration items.
- Building or providing specifications to build work products from the configuration management system.
- Maintaining the integrity of baselines.
- Providing accurate status and current configuration data to developers, end users, and customers.

The model describes the software engineering and management practices that characterize organizations as they mature their processes for developing and maintaining software. CMM consists of sets of recommended practices in a number of key process areas (KPA) that have been shown to enhance software process capability such as Requirements Management, Software Project Planning, Software Quality Assurance, Organization Process Definition, Training Programs and Integrated Software Management. [Paulk et al, 1993]

The model consists of five maturity levels and classifies software organizations according to their ability to control the elements mentioned above in each stage of the software development. For an organization to reach a higher level of maturity, it demands an extensive and coordinated effort by the entire organization. The five levels of CMM are described in the following subsections. Software Configuration Management is placed on level 2 on the maturity model, and is therefore considered to be essential to almost any software organization and product.

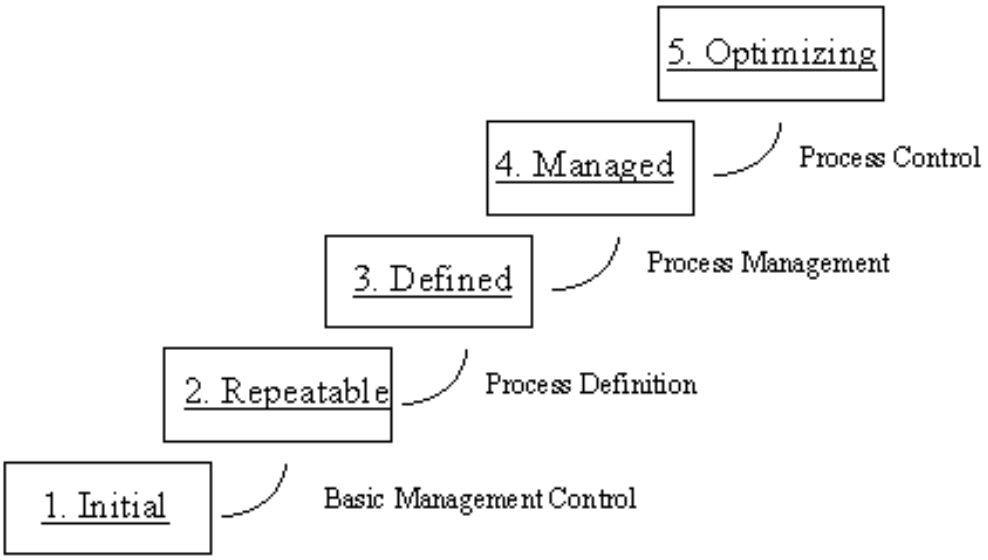

Figure 2.3-I. The five levels of Capability Maturity Model [CMM, 1993].

Level 1 - Initial (The "heroic" level)

> "The software process is characterized as ad hoc, and occasionally even chaotic. Few processes are defined, and success depends on individual effort." [CMM, 1993]

At the Initial level, the organization lacks a set of sound management practices. The organization depends upon the individual employee who assumes the role of the hero struggling to overcome all obstacles. As there is no documentation, projects at this level are often strongly severed or even forced to stop if the Project Manager decides to leave the project.

Level 2 - Repeatable (Level of Projects)

> "Basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications."[CMM, 1993]

When an organization is at the repeatable level, the software process is submitted to a basic form of control and therefore, it is possible to repeat the processes. The Project Manager can make reasonable estimates of the project and can measure and control the project on the basis of the project plans.

The systematizing is not yet well defined and exists mostly in the heads of the employees. If there is a moderate renewal in the staff, the Organization can still operate on the basis of the successful experiences of former projects.

Level 3 - Defined (The Organizational level)

> "The software process for both management and engineering activities is documented, standardized, and integrated into a standard software process for the organization. All projects use an approved, tailored version of the organization's standard software process for developing and maintaining software."[CMM, 1993]

At the defined level, the basic structure of the software processes are known and used in the organization. It is imperative that a standard process has been developed and that this process is well documented and institutionalized so that all employees are familiar with it.

The standard process is not static but is continuously improved on the basis of the experiences of the individual projects. To get to the third level, the organization needs to devote itself to development and the Management needs to dedicate the resources necessary to measure, evaluate and adjust the processes. Level 3 also demands that a group, called the Software Process Improvement Group (SPI), is responsible for all process activities and that a training program has been established.

Level 4 - Managed (The quantitative level)

> "Detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled."[CMM, 1993]

The processes are implemented in a way, which makes quantitative measurements possible at the fourth level. This enables the organization to create a set of quantitative standards for both processes and products. By knowing the usual status in each stage of products and processes, the projects can gain control of the products and processes because they can spot deviations instantaneously. Standards can be made in regard to the size of variations tolerated. Thus products are made predictable as processes operate within known boundaries.

Level 5 - Optimizing (Continuous improvements)

> "Continuous process improvement is enabled by quantitative feedback from the process and from piloting innovative ideas and technologies."[CMM, 1993]

At the highest level, process improvements are integrated into the working processes and function routinely. Focus is on continuous improvements and there is a possibility of spotting process weaknesses before they even occur, by analyzing data thoroughly and by improving processes continuously in order to avoid mistakes.

2.4. Software Configuration Management Models

Peter H. Feiler made a first approach to classify SCM functionality [Feiler, 1991]. He examined the software process as it is enforced by existing SCM systems and distinguishes four configuration management models each introducing specific functionality:

Checkin/Checkout Model. The basic SCM model introduces the concept of a repository holding multiple versions of a product component. Developers can copy versions from (checkout) and to (checkin) the repository.

Change-Oriented Model. As its name says, the Change-Oriented Model focuses on changes rather than on versions. In this model, versions are the product of a change set applied to a baseline. This model is useful for propagating and combining changes across users and sites.

Composition Model. The Composition Model extends SCM from the component level to the system level, introducing system models describing the system structure and configurations denoting versions of several components. Consistency issues are also found here.

Long Transaction Model. The Long Transaction Model introduces the notion of a workspace, where developers are isolated from each other's changes.

According to Dart [Dart, 1991], since Feiler's survey, many new SCM systems have emerged, and many have extended their initial functionality to incorporate functionality that was previously found in other SCM models. Although most of today's SCM systems are essentially based on one of these SCM model, a more fine-grained approach is required to capture the entire spectrum of functionality in SCM systems.

After Dart's study, a couple of new approaches have emerged. One of these experiments to integrate the four other models, or create an all-around model for SCM (which would cover all the functionality recognized in SCM), is Andreas Zeller's **Version Set Model** [Zeller, 1997], generally introduced in 'Version Set Model' Subsection 2.4.5.

2.4.1. Checkin/Checkout Model

In summary, the checkin/checkout model focuses on versioning of product components. The operational concepts of checkout, checkin, branch, and merge are low-level primitives for which users have to develop usage conventions to better address their SCM support needs. Examples of these low-level primitives are conventions for the use of branches, for maintenance of configuration information, and for supporting scopes of visibility of changes. [Feiler, 1991]

SCM systems primarily supporting this model focus on managing the repository. Support of users in their workspaces is limited to component branch locking in the repository as a means of coordinating modifications. In practice, users of such SCM systems have evolved conventions whose patterns emulate some of the concepts found in the other SCM models.

2.4.2. Change-Oriented Model

The change set model supports change-oriented configuration management in a natural way. It provides a link to change requests, which are a management tool for controlling the software process through change authorization and status tracking. It allows developers to view configurations in terms of collections of logical changes. Adding change sets to appropriate configurations propagates logical changes. Different integration strategies can be pursued for collections of concurrent logical changes by combining them in particular order. Appropriate queries can determine the degree to which logical changes have spread throughout a collection of system configurations being maintained. [Feiler, 1991]

Although change sets relate to transactions, they do not provide concurrency control. Because of this SCM systems do complement the change set model with the checkin /checkout model. Merge mechanisms necessary to support optimistic concurrency control are also applicable to determining and resolving change set conflicts.

2.4.3. Composition Model

The composition model operates on configurations by composing aggregates from components and selecting appropriate versions of each. The system structure is captured in a system model. The system model provides the link between configuration support, system build tools, and language systems. This link permits the SCM system supporting the composition model to include management of derived objects and checking of interfaces between components as well as between aggregates, i.e., subsystems. [Feiler, 1991]

SCM systems based on the composition concept evolve as follows. A system model and a selection rule identify a configuration in the repository that is the starting point for change. A developer defines a new configuration consisting of the system model and a selection rule that includes the workspace into which modified components are checked out. Once the work is completed, the developer preserves the configuration in the repository as a new system version by checking in all modified components as new versions. A new selection rule, which does not include the workspace, but refers to the new component versions, must be created to enable the developer to identify the same system configuration after it is released from the workspace. [Feiler, 1991]

Selection rules provide guidelines for the SCM system to perform version selection. This allows the developer to express selection of alternatives in a natural way. Support for evolution must be expressed in terms of composition with changing selection rules. Support for change migration is limited to the capabilities of change merging at the component level and record keeping by the developer.

2.4.4. Long Transaction Model

The long transaction model supports evolution at the configuration, i.e., composite level in a natural way. It provides stable workspaces with control over isolation from external change, scopes of visibility for changes, and coordination of concurrent change activities at various granularities of the system structure. By managing workspaces, CM systems can support developers during active development. A range of concurrency control schemes can be attached to transactions. Since each transaction has a different impact on cooperative team support, it is desirable for a SCM system to support the adaptation of schemes to different process needs. [Feiler, 1991]

When used as the primary CM model in a CM system, long transactions play the role of development paths in the repository as well. Change propagation restrictions and limitations in change query capabilities are common. However, information flow restrictions due to concurrency control schemes are often too restrictive for change propagation in the repository. Long transactions do not directly support composition, but support evolution of subsystems based on a decomposition according to the system structure. System families are supported as independent development paths treated as system variants, or through explicit variant support within a workspace.

Long transactions represent a working context for logical changes. The modifications recorded as a part of the transaction log represent a logical change. Such information is also the basis of the change-oriented model discussed in one of the previous subsections.

2.4.5. Version Set Model

In his 320 page thesis work, Andreas Zeller not only covers almost the whole SCM scheme, but he also attempts to provide a common formal and adaptive base for the technical aspects of software configuration management. As the base for this integration Zeller has chosen feature logic, a logic denoting objects by specifying their possible attributes (or non-attributes). Characterizing objects by their features is a common technique in SCM, so Zeller's choice seems natural to me.

Using feature logic, Zeller's thesis models and integrates common SCM functionality such as attributed components, repositories, work spaces, variant sets, revision histories, and consistency checking in a single concept, called version set. Version set gathers versions, components, and configurations by their features. SCM functionality is realized through set operations. Versions are selected and refined through set intersection. Set union realizes the grouping of versions to repositories. Subsumption and disjointness express inclusion and exclusion of changes, structuring the version space.

Version sets do not introduce new concepts into SCM; instead, they expose new ways of combining and integrating existing concepts and thus provide much more flexibility in adapting SCM systems to their users. In short, Zeller has designed the version set model as an attempt to integrate the current spectrum of SCM functionality into a single, hopefully simple and elegant formalism, allowing for adaptive combinations of SCM concepts with predictable effects. [Zeller, 1997]

3. Goals for SCM

No matter how large the project may be the software configuration management has a critical effect on quality. Good software configuration management is essential for efficient development and maintenance, and for ensuring that the integrity of the software is never compromised. Bad software configuration management can paralyze a project. Sensible change requests may fail to be approved because of fears that they cannot be implemented correctly and will degrade the system.

Effective change requires an understanding of the current process and a goal. Defining goals for the software process can be seen as a part of the first step in software process improvement (Step 1 in the Figure 3-I). [SPICE]

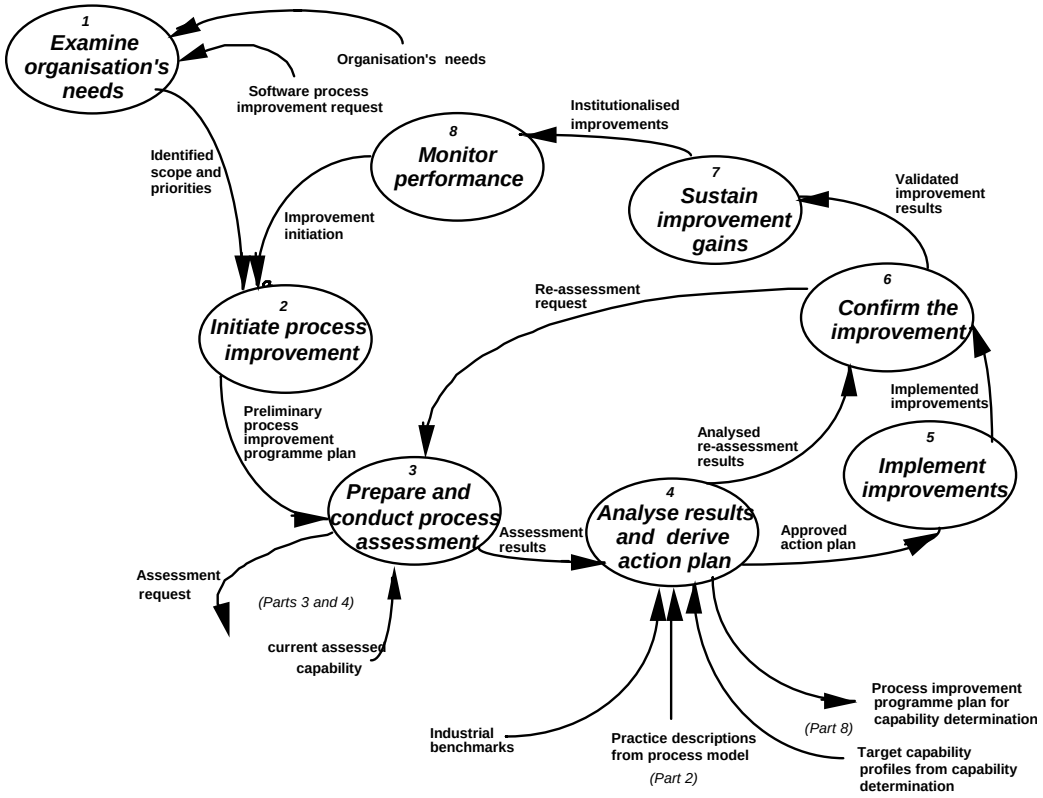

Figure 3-I. Software process improvement steps. [SPICE]

Many of the aspects of SCM have long been considered to be overly formal and bureaucratic. The developers blame the SCM procedures to be slowing them down and decreasing their creativity. At the same time, people in charge for SCM consider developers to be ignoring the concerns of SCM. This kind of situation is a flaw in the process, and should be acted upon. It is important that the SCM solution recognizes the problem and tries to overcome the “wall” between the developers and software configuration managers.

The standards have also recognized this danger of following the given guidelines too literally. Every SCM organization should keep this in mind, and take the goals given by CMM should be considered to be suggestions, not the only way to do things.

3.1. Using standards in goal definition

The leading quality standards like SEI CMM and ISO 9001 provide fine-tuned goals and practices; as many previous papers have recognized this, it is justified to use them as a base for goal definition. The purpose of the standards is to ensure software quality and cover the software organization as a whole. Anyone in control of a software organization or product should at least get to know the basic features of these standards.

The SEI has associated eighteen key process areas (KPA) with each levels of the maturity model. The KPAs describe those software engineering functions that must be present to satisfy good practice at a particular level. The maturity levels are additive (Level 3 contains all Level 2 KPAs and those noted for Level 2). [Pressman, 1997][CMM, 1993]

Process Maturity Level 2

- Software configuration management
- Software quality assurance
- Software subcontract management
- Software project tracking and oversight
- Software project planning
- Requirements management.

Process Maturity Level 3

- Peer reviews
- Intergroup coordination
- Software product engineering
- Integrated software management
- Training programme
- Organization process definition
- Organization process focus

Process Maturity Level 4

- Software quality management
- Quantitative process management

Process Maturity Level 5

- Process change management
- Technology change management
- Defect prevention

Each KPA is described by identifying the following characteristics:

- Goals – the overall objectives that a KPA must achieve.
- Commitments – requirements (imposed by the organization) that must be met to achieve goals, provide proof of intent to comply with the goals.
- Abilities – those things that must be in place (organizationally and technically) that will enable the organization to meet the commitments.
- Activities – Specific tasks that are required to achieve the KPA function.
- Methods for monitoring implementation – the manner in which the activities are monitored as they are put in place.
- Methods for verifying implementation – the manner in which proper practice for the KPA can be verified.

[Pressman, 1997][Paulk et al, 1993]

3.1.1. Goals for SCM in CMM

> ”The purpose of Software Configuration Management is to establish and maintain the integrity of the products of the software project throughout the project’s software life cycle. Software Configuration Management is an integral part of most software engineering and management processes.”[CMM, 1993]

The overall goals for SCM that CMM defines are [Paulk et al, 1993]:

Goal 1 Software configuration management activities are planned.

Goal 2 Selected software work products are identified, controlled, and available.

Goal 3 Changes to identified software work products are controlled.

Goal 4 Affected groups and individuals are informed of the status and content of software baselines.

To reach the overall goals, CMM also defines the following generic goals (GC) which same for all key process areas):

1) Achieve specific goals (SG)
2) Institutionalize a Managed Process
3) Institutionalize a Defined Process
4) Institutionalize a Quantitatively Managed Process
5) Institutionalize an Optimizing Process

The specific goals (SG) for SCM are:

1) Establish Baselines
2) Track and Control Changes
3) Establish Integrity

CMM also describes specific practices (SP) to both specific and generic goals:

SG 1 Establish Baselines

SP 1.1-1 Identify Configuration Items

SP 1.2-1 Establish a Configuration Management System

SP 1.3-1 Create or Release Baselines

SG 2 Track and Control Changes

SP 2.1-1 Track Change Requests

SP 2.2-1 Control Configuration Items

SG 3 Establish Integrity

SP 3.1-1 Establish Configuration Management Records

SP 3.2-1 Perform Configuration Audits

GG 1 Achieve Specific Goals

GP 1.1 Perform Base Practices

GG 2 Institutionalize a Managed Process

GP 2.1 Establish an Organizational Policy

GP 2.2 Plan the Process

GP 2.3 Provide Resources

GP 2.4 Assign Responsibility

GP 2.5 Train People

GP 2.6 Manage Configurations

GP 2.7 Identify and Involve Relevant Stakeholders

GP 2.8 Monitor and Control the Process

GP 2.9 Objectively Evaluate Adherence

GP 2.10 Review Status with Higher Level Management

GG 3 Institutionalize a Defined Process

GP 3.1 Establish a Defined Process

GP 3.2 Collect Improvement Information

GG 4 Institutionalize a Quantitatively Managed Process

GP 4.1 Establish Quantitative Objectives for the Process

GP 4.2 Stabilize Subprocess Performance

GG 5 Institutionalize an Optimizing Process

GP 5.1 Ensure Continuous Process Improvement

GP 5.2 Correct Root Causes of Problems

3.2. Goals for the organization

Most of the previous studies of SCM evaluation have concentrated on tools, but the generic (organizational) goals / requirements for the organization should not be overlooked. An organization may have excellent tools to carry out an efficient SCM process, but the lack of organizational methodology and training can prevent the efficient use of those tools. Although this can be applied the other way around as well, I see the organizational process development at least equally important to the efficiency of the tools to support the process. The goals set for the SCM by CMM (Subsection 3.1.1.) provide a basis for the organizational requirements [Paulk et al, 1993].

3.3. Requirements for the SCM tools

The base practices for SCM, defined by CMM, give the minimum of what should be done in a SCM process. These specific goals and practices (presented in Subsection 3.1.1.) will be mostly accomplished with the support of tools.

Summarizing the above, SCM is the practice of:

- Tracking changes made to all components of a software project;
- Recording the way the components form a release;
- Providing controlled access to current and previous software versions;
- Enforcing site-specific development policies.

Turning the base practices to a software development environment, a comprehensive SCM product enables developers to ensure the accuracy of releases, develop and build software in parallel as a team, isolate specific files relevant to given tasks, manage multiple workspaces, and reproduce specific releases from the past. As noted in definitions, SCM demands the tools to offer a lot more than just traditional version control tools. Combining these, a comprehensive SCM solution therefore integrates four key functions:

- Version control
- Workspace management
- Build management
- Process control

SCM solution and tools can be evaluated by categorizing the evaluation according to the key functions mentioned above. The goals for the (SCM) tools mentioned in the four key areas should be adapted and integrated with the organization's development environment and policies.

The following Subsections (3.3.1. -3.3.5.) define the goals for SCM tools, and can be used as evaluation checklists.

3.3.1. Version control

- Control the versions of all files and system objects, including directories
- Store all versions efficiently
- Automatically log all changes
- Support parallel development through rich branching, labeling, and merging facilities
- Merge files safely and automatically
- Merge directories
- Provide a highly reliable, recoverable data storage

3.3.2. Workspace management

- Provide transparent access to versions
- Provide full SCM functionality in the developer's native work environment
- Support developers working from home or remote locations.
- Create multiple workspaces
- Dynamically evaluate configurations
- Support arbitrary, ad-hoc configurations and unforeseen necessities

3.3.3. Build management

- Support make-compatible building
    - Sw build procedures are described in standard Unix makefile)
- Automatically detect source and header file dependencies
- Document exactly what went into every build
- Guarantee build reproducibility
- Share derived objects automatically
- Perform distributed parallel building, with load balancing

3.3.4. Process control

- Support leading quality standards
- Support project and enterprise specific development policies
- Support tight integration with change request management tools
- Allow tracking of state transitions
- Trace requirements
- Support flexible locks and access controls.
- Allow easy creation of pre- and post-event triggers.

3.3.5. Architectural and organizations requirements

- Integrate with existing tools and development environments
- Scale to support organization growth
- Offer both command-line and graphical user interfaces
- Support development at multiple sites
- Support off site (work-at-home) development
- Interoperate across various operating systems / environments

3.4. Goals for agile SCM

I think that there is one issue that all SCM organizations have in common: The challenge of finding the right amount of control. How to control the change effectively, but at the same time keep the process from getting too formal and bureaucratic? I believe that the solution could come from making the SCM as practical and agile as possible.

Agile Values

We are uncovering better ways of developing software by doing it and helping others do it.
Through this work we have come to value:
- **Individuals and Interactions** over Processes and Tools
- **Working Software** over Comprehensive Documentation
  - **Customer Collaboration** over Contract Negotiation
  - **Responding to Change** over Following a Plan
That is, while there is value in the items on the right, we value the items on the left more.

Figure 3.4-I. Agile Values from “Manifesto for agile software development” [Martin, 2002]

The values presented in Figure 3.4-I offer the needed guidance towards agile SCM. The overall efficiency will benefit from more practical and agile SCM, so these values should be considered as a part of the goal definition phase.

3.5. SCM goals summarized

Here I present a summary of the important goals for SCM:

(1) SCM activities are carried out to ensure the following:

- Software components can be identified
- Software is built from a consistent set of components
- Software Components are under version control, and are available and accessible
- Software components never get lost (e.g. after media failure or operator error)
- Every change to the software is approved and documented
- Changes do not get lost (e.g. through simultaneous updates)
- It is always possible to go back to a previous version (baselines are established)
- A history of changes is kept, so that is always possible to discover who did what and when.

(2) The activities are defined in the SCM plan

(3) Project management is responsible for organizing software configuration management activities, defining software configuration management roles (e.g. configuration coordinator, software builder, etc.), and allocating staff to those roles.

(4) Development personnel share items safely and efficiently. Project management requires accurate identification of all items, and their status, to control and monitor progress. Quality assurance personnel need to be able to trace the derivation of each item and establish the completeness and correctness of each configuration. The configuration management system provides visibility of the product to everyone.

(5) SCM tools provide methods to meet the goals in an efficient way. Tools should assist the SCM related work on all levels of the organization.

4. Evaluating the SCM Process

> "It is important to be able to show the leaders the benefits that modern SCM will provide and the problems that may be encountered if a consistent approach to SCM is not used on a program. It is also imperative to set up SCM tools with the cooperation of software developers. If management and software developers do not buy into the SCM process and its procedures, then the process is doomed to failure. A team approach, with inputs from program management to software engineering, to software quality organizations is most useful. Finally, educating the users in the use of SCM tools is essential for a smoothly running process"
>
> - Headock, Rita [Mosley et al, 1996]

An efficient SCM process does not exist without the organizational policies and methods, and the right tools to support those methods. As the organizational methods and the tools can be seen as equally important to the overall performance of the SCM process, both should be evaluated separately. Evaluating the SCM process in the organization should be started from the organizational environment and methods. The support of tools to the overall process can then be evaluated more efficiently. The defined goals (Chapter 3) should be used when conducting the evaluation.

Cost efficiency is also very important to all organizations, so it should be taken as criteria when conducting the evaluation. In general, a smoothly running SCM process will itself save the organization from many unnecessary costs, but to verify this by evaluation, some pointers can be raised from the previous studies concerning the industry. The Westinghouse Top 10 Common "cents" Rules have successfully been used to evaluate and acquire SCM toolsmaking sense and saving money [Mosley, 1995][Mosley et al, 1996]:

1) Haste makes waste.
2) Practitioners are prime evaluators.
3) Evaluate against user's needs.
4) Don't believe it unless you see it.
5) Don't buy it until you've tried it.
6) Pay now or pay later.
7) Everything is negotiable.
8) Acquire only what is needed.
9) Create a champion.
10) Be constructive, not destructive.

The needs and business goals of the organization determine the software process improvement goals that help to identify improvement actions and their priorities. Software process improvement is accomplished in a series of steps or specific improvement actions such as introducing new or changed practices into software processes or removing old ones. Like defining goals for the process, software evaluation can be seen as one of the initial steps in software process improvement (step 3 in the Figure 3-I).

The basic principles of software process improvement described by SPICE are [SPICE]:

- Software process improvement is conducted on the basis of process assessment and process effectiveness measures.
- Software process assessment produces a current process capability profile, which may be compared with a target profile based on the organization's needs and business goals.
- Process effectiveness measures relate the identification and priorities of improvement actions to the organization's needs and business goals, and achievement of software process goals.
- Software process improvement is a continuous process. Improvement goals identified and agreed within the organization are realized through a process improvement process that continues through multiple cycles of planning, implementing and monitoring activities.
- Improvement actions identified within a process improvement process are implemented as process improvement projects.
- Metrics are used for monitoring the improvement process in order to indicate progress and to make necessary adjustments.
- Software process assessment may be repeated in order to confirm that the improvements have been achieved.
- Mitigation of risk is a component of process improvement and should be addressed from two viewpoints:
  - The risk inherent in the current situation, and
  - The risk of failure in the improvement initiative.

4.1. Evaluating the organization

Organizational evaluation emphasizes on the SCM process and the people:

- Are the organizational policies and procedures well defined, and informed throughout the organization?
- Is the process carefully planned and documented (in the SCM Plan)
- Are there enough resources to accomplish the goals set in the plan?
- Are responsibilities assigned and well known to all?
- Are the people motivated throughout the organization to work towards the common goal and carry out the SCM activities?
- Are the training needs recognized and acted upon?

The SCM process should be continuously evaluated and improved:

- Is the process monitored and controlled?
- Is the process continuously improved?
    - Is improvement information collected?
    - Can the root causes of problems be recognized and corrected?
- How do the process and the organization manage configurations?
- What is the status of the different levels of management?
- Is the process defined and documented in the SCM Plan established, as it should be?
- Is the organization continuously looking for more efficient methods and procedures to automate the process?

Problem tracking and change management is an essential part of SCM, and should provide the possibility to generate reports and metrics to examine the process and the product(s). For example, such metrics include the:

- Number of defects found after releases,
- Number of change requests,
- Number of bugs found during the development,
- Number of bugs found during testing,
- Time to identify and correct bugs,
- Number of source lines delivered, and
- Number of reused lines of source code.

Most of the metrics are used recognize the improvement areas in the SCM and in the overall software development process. For example, if the time to correct the identified bugs is very long, the bug reporting and correcting procedure might require some modification or perhaps the people correcting the bugs need more attention and motivation by the management.

The increase in the quality of the final product is usually one of the main goals for SCM. Number of defects found after releases is probably the most important factor when considering the quality of the product. In products where defects in the final release are totally unacceptable (for example, software for space shuttles), the amount of control and testing must be increased to minimize or remove all the defects from the final product.

4.2. Evaluating the tools for SCM

> "To achieve a successful evaluation and selection of the ideal tool(s), the evaluator should first understand the basic configuration management definitions, which are the same for hardware and software. Evaluator should also understand the intent and purpose of the SCM process, its current status, and the attributes necessary to successfully perform the process on a project or within developing organization." [Berlack, 1995]

Although tools alone will not solve an organization's SCM problems, the purpose of tools is to reduce the workload and to reduce the need of resources. The defined goals provide a basis for the evaluation.

The goal for a successful evaluation and selection program is to produce a tool that:

- Will do what the organization wants it to do.
- Fits the job and the current development environment.
- Is cost effective and will reduce the workload.
- Will have adequate vendor support and little maintenance.

The SCM requirements should be carefully considered when evaluating the tools. At any given time, SCM must know what version of each software configuration item has been delivered to the customer. If a SCI is changed, SCM must know what other SCIs this change will have impact. SCM must control the change of every SCI, and a SCM process to manage the change must be defined.

Summarizing the above: SCM tools should provide help when asking these very important questions for SCM to answer [Mosley et al, 1996]:

- What will be delivered or have been delivered to the customer?
- If the change has to be made, what will it impact?
- Since the change is inevitable, how will it be managed and controlled?

4.3. Guidelines for Evaluation

(1) The purpose and/or goals of the SCM process must be known before a meaningful evaluation can be done: The purpose of SCM is to plan, organize, control and coordinate the identification, storage and change of software through development, integration and transfer. Every project must establish a software configuration management system. All software items, for example documentation, source code, executable code, files, tools, test software and data, must be subjected to SCM.

(2) The evaluation team consists of personnel from various parts of the development organization. Evaluation will be more effective if users from various areas (e.g. developers, testers, quality assurance people, usability experts, technical leaders, build managers, project managers, etc.) bring their knowledge and insight to the evaluation team.

(3) The evaluation steps are documented in an evaluation plan. Planning and documenting the evaluation ensures that the results can be used properly. The documented process can then be supervised and controlled. The evaluation process and plan can then be verified, and the evaluation process can be improved in the future.

5. SCM Evaluation Plan

SCM can be seen as a critical part on any organization, and the process should always be under evaluation and improvement. With the information on this thesis, a combination of standards, goal definition, and evaluation, the organization can recognize and prioritize the improvement areas in the process (Figure 5-I below).

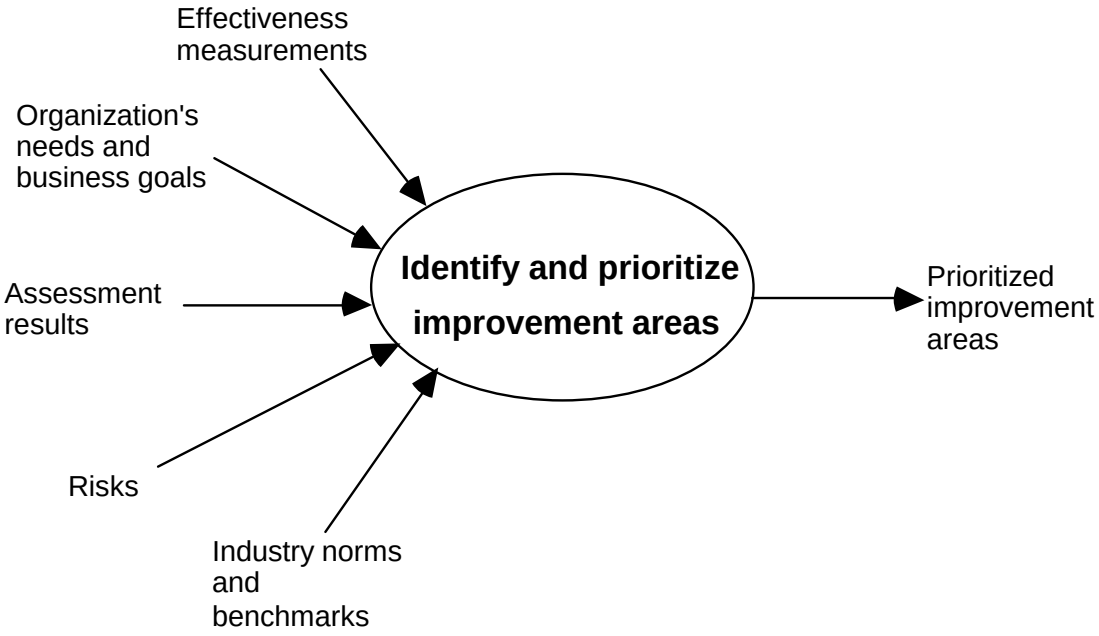

Figure 5-I. Identifying and prioritizing the improvement areas. [SPICE]

The different standards and guidelines already offer proven basis to evaluate and improve software configuration management. However, all of the standards and the studies give a different aspect that the others do not. Because of this, I combine the practices from the standards and previous studies into one set of SCM evaluation and improvement phases (Subsection 5.2.), which can be used as a basis for the software configuration management evaluation plan. To implement the evaluation plan, I form guidelines on what to consider when conducting the evaluation for SCM (Section 5.1.)

5.1. Conducting the evaluation for SCM

The evaluation is divided into two parts: organizational evaluation and tool evaluation. These two are not necessarily divided during the evaluation process, but are covered separately in this paper to clarify the criteria of both to maximum.

The main purpose of the organizational evaluation is to find the source that causes the possible problems. In addition to this, the evaluation is used to find the areas for improvement. The first part of the organizational evaluation focuses on the methods, procedures and policies used in the organization, and the consistency of those against the higher-level goals and requirements. The other part of the organizational evaluation is to chart the know-how of the people working in the SCM organization. The improvement of the organization can then begin by creating an improvement plan based on the evaluation. If training needs are recognized during the evaluation, a separate plan about training people should also be formed or included in an appropriate existing plan. To accomplish the organizational requirements, methods and tools should be evaluated not only by predefined criteria, but also by how they can be used to make the process more efficient or perhaps completely remove problems.

Therefore the evaluation of tools is ideally intended to find a SCM tool that can automate most of the SCM solution. The keys to success for evaluation are having properly skilled evaluation team, designing useful evaluation criteria, having realistic and comprehensive set of requirements, and running meaningful evaluation tests based on the criteria. According to Dart [Dart, 1996], in a good evaluation process:

- An evaluation team is chosen.
- An evaluation methodology and selection criteria are designed.
- A set of requirements, categorized by priority, is developed.
- All possible sources of information on SCM tools, from trade magazines to analysis companies are examined.
- An initial set of candidate tools is examined.
- Two final candidates are chosen.
- Detailed evaluation tests, e.g. proof of concept, are run.
- Reference companies that already use those tools are contacted.
- Based on the evaluation criteria, the better candidate is chosen.

5.1.1. Team – The Champion

The key to the most effective ways to evaluate the current practices and to transition new technology into use is to find a well-respected potential user of the technology. First the user is trained on the process and the technology. The same user then evaluates the process and technology. The champion becomes the expert user, facilitator, and trainer of the tool. [Mosley et al, 1996]

Because SCM integrates the work created by many (if not all) people throughout an organization, the person(s) in charge of SCM needs a broad understanding of software engineering principles and the cultural aspects of the organization. [Kinsbury, 1996]

Finding potential key user(s) to conduct the evaluation, improvement and adaptation starts the evaluation and improvement implementation. One user or group could become the SCM expert(s), but the evaluation team should consist of various representatives of the user community. The team can include developers, testers, quality assurance people, usability experts, technical leaders, build managers, project managers, etc. All provide their perspectives and ensure that their needs are addressed, while providing their own experiences, skill set, and processes to address three important areas apart from functionality requirements: usability, performance, and scalability requirements. [Dart, 1996]

5.2. SCM Evaluation and Improvement plan

A tool alone will not solve an organization's SCM problems. Other technical and cultural issues must be addressed. A complete understanding of SCM fundamentals and the necessary policies, procedures, and instructions must be in place to perform the SCM process prior to setting up an automated process. [Berlack, 1995]

The phases presented in a paper by Kinsbury [Kinsbury, 1996] form a Road Map towards adopting the SCM technology. The phases follow and complement the risk-based adoption methodology presented by Dart [Dart, 1996]. The road map created with the phases provides structured guidance, identifies tasks, and addresses the complexities involved with SCM technology adoption. The following Subsections (from 5.2.1 to 5.2.6.) present the phases of the evaluation.

At all phases, it is important to reinforce management's commitment to the adoption effort and to provide training. Training becomes a critical part of ensuring that SCM principles are adopted by the organization.

5.2.1. Preparation and Planning

Start by creating the SCM adoption team. The adoption team is responsible for implementing the adoption strategy and plays an important role in the adoption effort. The team monitors and participates in all phases of the adoption effort. Members of the adoption team typically include:

- A leader who is responsible for the adoption effort
- A sponsor who has the authority to empower the team and provide the support required to tackle the difficult SCM problems
- A champion or technical experts who understand the technology (and SCM process)
- Representatives from user community

The team begins by developing the SCM Adoption Plan. The plan details the benefits of SCM, outlines the adoption schedule, defines the policies and procedures involved in the adoption effort, establishes success criteria, and establishes roles of the adoption team.

Then the requirements are defined and prioritized. Clear understanding of the organization's strategic goals is required to evaluate the SCM requirements. All levels of development, therefore all people affected by SCM, must be surveyed to identify their SCM requirements and to determine their roles in the SCM process. Training requirement of all people affected by the SCM tool, process, and procedures should be given special attention.

All levels of management must be aware of the SCM problems the organization is currently encountering. The benefits of implementing a SCM solution must also be presented. This involves presenting the financial and scheduling advantages of the SCM solution, e.g. increase in programmer productivity by automating SCM tasks, less rework, and support for reuse.

A risk management plan is also developed. The risk management plan identifies risks that could affect the outcome of the adoption effort, e.g. changing over to a new operating system during the adoption effort. The adoption team is responsible for identifying and addressing the risks throughout the project.

5.2.2. Process Definition

The goals of this phase are to define the current SCM process, evaluate the process, and define a new, improved process if required. The process is then analyzed to identify which areas would benefit from automation. The defined process is pertinent to the successful implementation of SCM. Without defined process, the organization will make little progress in the adoption effort.

5.2.3. Tool Evaluation

The goal of this phase is to select a tool that satisfies the organization's SCM tool requirements.

The guidance on tool evaluation and selection is found in Chapter 4 (4.2. Evaluating tools for SCM), and a more detailed information can be found in referenced material [Berlack, 1995], [Mosley, 1995], and [Mosley et al, 1996].

5.2.4. Pilot Project Implementation

The purpose of this phase is to determine how well the SCM tool, process, and procedures satisfy the organization's requirements. A pilot project allows the tool's functionality to be tested with real data on a real project. The pilot also allows the methods and the procedures to be prototyped and provides feedback on how the users respond to the tool.

It is important to select a pilot that will address the high-risk areas, but not affect the project's critical path. The adoption team develops standards, policies, and procedures as well as ensures that users are trained to perform their SCM duties. Successes and failures are documented and compared to the success criteria identified in the adoption plan.

5.2.5. Rollout to Other Projects

This phase involves incrementally migrating the processes, procedures, and the tools to other projects. Training and dealing with resistance to change are key activities in this phase. The SCM tool, process, procedures, and training needs are examined and adapted for each project. The adoption team implements, evaluates, and monitors rollout activities. This phase is complete when the SCM tools, processes, and procedures are routinely used on all selected projects.

5.2.6. Process Improvement Phase

This phase involved evaluating adoption activities, capturing strategies that worked, and making recommendations for process improvements. The adoption team has developed technology transition skills that can be applied to other types of technology adoption, and the organization has reaped the benefits of automated SCM support. At this point, new problems will arise, i.e. company reorganizations, which require the adoption process to be initiated again.

5.3. Evaluation Plan Guidelines

(1) Whether one's purpose is to evaluate and improve the existing SCM process, or to adopt new SCM technology into organization, the process should follow the phases presented above.

(2) Evaluation process and plan should be "under configuration control". Evaluation process and plan can then also be reviewed and improved.

(3) To continuously improve the process, initiate the first phase of the evaluation and improvement plan after the last phase is finished.

(4) Always start by analyzing the results of the previous evaluation/ improvement cycle.

6. Discussion

SCM is an important part of software engineering, and getting familiar with it will help one to see how important management of change and configuration can be. However, as companies can have specific SCM goals, different individuals and organizations should tailor the standards and activities described in this chapter to their needs. For example, the software configuration management can be implemented as a part of quality assurance without any user or group responsible for the activities, or as a separate unit, which will monitor the whole process.

Summarizing this thesis, the minimum activities that should be done in the beginning of the SCM process are (and in running SCM processes the existence of these should be confirmed by evaluation):

- Identify the SCM activities to be implemented.
- Identify and choose the models and methods.
- Set the user roles responsible for different activities.
- Make the schedule (can be expressed as absolute dates, dates relative to activities, project milestones or simple sequence of events).
- Find out what tools, techniques and equipment are to be used.
- Identify the need for resources (personnel and training, in addition to the tools, techniques and equipment mentioned above).
- Document all of the above into the SCM plan.

The software configuration management starts at the very beginning of the software project. The methods to be used for the identification of configuration items, control and implementation of change, and also recording and reporting change implementation status should be documented in a software configuration management plan.

The SCM plan [IEEE, 1990] documents what SCM activities are to be done (identification of activities and models), how they are to be done (identification methods, policies and procedures to be used), who is responsible for doing specific activities (setting the user roles), when they are to happen (scheduling) and what resources are required (identification of tools, techniques, equipment, personnel and training needed). Efficient implementation of the documented SCM activities will most likely result in better quality in the final product, which is also carefully documented, tested and easy to develop further.

I raise a few issues from this paper, the standards and previous studies; and these should be given careful attention when adopting SCM technology or evaluating and improving the existing SCM process:

- **Define SCM process, procedures and policies**; the whole SCM process is defined and documented to the SCM Plan. In addition to this thesis, standards and previous studies offer detailed information on the SCM process [IEEE, 1990], [Paulk et al, 1993], and [Dart, 1996].
- **Define realistic schedule**; schedules ease the tracking of the current status in the project. Releases and milestones attached to those also help.
- **Carefully evaluate and choose the best possible SCM tools**; the purpose of the tools is to make the SCM process more efficient. Efficient tools reduce the workload by minimizing the manual work, and therefore reduce the number of defects caused by human errors.
- **Put emphasis on people**; a tool alone will not solve an organization's SCM problems. Everyone in the organization must understand the SCM process and therefore know where he or she stands in the SCM process.
- **Use experts**; the person(s) in charge of SCM needs a broad understanding of software engineering principles and the cultural aspects of the organization.
- **Assign responsibilities**; all activities defined in the SCM process are assigned throughout the SCM. Everyone in the organization must know what he or she should do.
- **Train people**; training requirements of all people affected by the SCM tool, process, and procedures must be identified.
- **Keep everybody informed of changes**; everyone affected by SCM must be kept informed of the changes to SCM process.

6.1. Recommendations and restrictions

The implementation of the SCM varies in every organization, so it is nearly impossible to give specific instructions on how to perform SCM. Therefore the guidelines in this thesis and in the industry standards must be understood as guidance to the right direction, not as specific instructions.

The results and conclusions provide an extensive summary of SCM, but to cover the whole process area, a much larger study would be needed. For this reason, the lists and guidelines presented here cannot be generalized, although the conclusions in this thesis are based on established standards and recognized papers written by respected authors. Also no evaluation or study of implementation was made as a part this thesis.

However, the overview given here gives introduction to the process and hopefully some ideas on how to adapt SCM in research or industry. And the combination of the whole thesis, the SCM evaluation and improvement plan can easily be tailored to fit any organization's needs. As an example, the evaluation and improvement practices can be adapted and used in distributed (multiple sites) organization.

6.2. Future research

To find out the best possible practices and methods, future studies should apply the evaluation steps given here and in related work to real life solutions of SCM. A comprehensive summary or study to cover the whole SCM process area could also be useful.

One interesting area to study would be to find out how the changes in organizational structures will affect the software configuration management and the software engineering process in general. This is especially important as the use of subcontractors and distributed organization models are becoming more and more common.

References

[Basili et al, 1994] Basili, V.R., Caldiera G., Rombach, H.D.. “The Goal Question Metric Approach”. In: Encyclopedia of Software Engineering, John Wiley&Sons,1994. <http://www.cs.toronto.edu/~sme/CSC444F/handouts/GQM-paper.pdf> March 2002.

[Berlack, 1992] Berlack, H. Ronald. “Software Configuration Management”. John Wiley & Sons, Inc., 1992.

[Berlack, 1995] Berlack, H. Ronald. “Evaluation and Selection of Automated Configuration Management Tools”. November/December 1995.

[Clemm, 1989] Clemm, G. M.. ”Replacing Version Control with Job Control”. Proceedings 2nd Intl. Workshop on Software Configuration Management, ACM, Princeton, 1989.

[CMM, 1993] Paulk, Mark C., Curtis, B., Chrissis, Mary B., and Weber Charles V.. “Capability Maturity Model for Software, Version 1.1”. Software Engineering Institute, Carnegie Mellon University, 1993.

[Dart, 1991] Dart, Susan. “Concepts in configuration management systems”. In: Proceedings of the 3rd International Workshop on Software Configuration Management,1-18. ACM, Princeton, 1991. <http://www.sei.cmu.edu/legacy/scm/abstracts/abscm_concepts.html>. March 2002.

[Dart, 1996] Dart, Susan. “Achieving the Best Possible Configuration Management Solution”. Crosstalk, September 1996. <http://www.stsc.hill.af.mil/crosstalk/frames.asp?uri=1996/09/achievin.asp> March 2003

[Feiler, 1991] Feiler, Peter H.. “Configuration management models in commercial environments”. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1991. <http://www.sei.cmu.edu/legacy/scm/abstracts/abscm_models_TR07_91.html> March 2002.

[IEEE, 1990] “IEEE Standard Glossary of Software Engineering Terminology”, The Institute of Electrical and Electronics Engineers, 1990.

[IEEE, 1987] “IEEE Guide to Software Configuration Management”, The Institute of Electrical and Electronics Engineers, 1987.

[ISO 9000-3] “Quality management and quality assurance standards – Part 3: Guidelines for the application of ISO 9001 to the development, supply and maintenance of software”. International Standards Organization, Geneva, Switzerland, 1991.

[ISO 15504] “ISO/IEC 15504 - An Emerging Standard on Software Process Assessment”. International Standards Organization, Geneva, Switzerland, 1998.

[Järvinen, 2000] Järvinen, P., Järvinen, A. ”Tutkimustyön metodeista (On Research Methods)”. Opinpaja Oy, 2000.

[Kinsbury, 1996] Kinsbury, Julie. “Adopting SCM Technology”. Crosstalk, STSC, Hill Air Force Base, Utah, March 1996.

[Martin, 2002] Martin, Robert C.. ”The Agile Manifesto”, also appearing in Chapter 1: Agile Practices, pp. 3-11 of Agile Software Development: Principles, Patterns, and Practices; Addison-Wesley, November 2002.

[Mosley, 1995] Mosley, V.. “Improving Your Process for the Evaluation and Selection of Tools and Environments”. Crosstalk, STSC, Hill Air force Base, Utah, September 1995. <http://www.stsc.hill.af.mil/crosstalk/frames.asp?uri=1995/09/Improvin.asp> March 2003.

[Mosley et al, 1996] Mosley V., Brewer F., Headock R., Johnson P., LaBarre G., Mazz V., and Smith T.. “Software Configuration Management Tools: Getting Bigger, Better and Bolder”. Crosstalk, STSC, Hill Air force Base, Utah, January 1996.

[Paulk et al, 1993] Paulk, Mark C., Weber, Charles V., Garcia, Suzanne M., Chrissis, Mary B., and Bush, Marilyn W., “Key Practices of the Capability Maturity Model, Version 1.1”, Software Engineering Institute, Carnegie Mellon University, 1993.

[Pressman, 1997] Pressman, Roger S.. “Chapter 9 (Software Configuration Management)”. In: Software Engineering A Practitioner's Approach, 223-238. McGraw-Hill Companies, 1997.

[Rahikkala, 2000] Rahikkala, Tua. ”Towards virtual software configuration management, A Case study”. VTT Electronics, Oulu, Finland. VTT Publications 409, 2000. <http://www.inf.vtt.fi/pdf/publications/2000/P409.pdf > March 2002.

[SPICE] “SPICE - Software Process Improvement and Capability determination” <http://ww.soi.qu.edu.au/spice>.

[Taramaa, 1998] Taramaa, Jorma. ”Practical development of software configuration management for embedded systems”. VTT Electronics, Oulu, Finland. VTT Publications 366, 1998. <http://www.inf.vtt.fi/pdf/publications/1998/P366.pdf> March 2002.

[Zeller, 1997] Zeller, Andreas. “Configuration Management with Version Sets: A Unified Software Versioning Model and its Applications”. Ph.D. thesis, Technische Universität Braunschweig, April 1997. <http://www.cs.tu-bs.de/softech/papers/zeller-phd/> March 2002.

Web resources

The ACME Project (Assembling Configuration Management Environments for Software Development).
http://www.cmcrossroads.com/bradapp/acme/

Bibliography on Software Configuration Management.
http://liinwww.ira.uka.de/bibliography/SE/scm.html

CM Resource Guide.
http://www.cmiiug.com/Sites.htm

CM Today.
http://www.cmtoday.com/yp/configuration_management.html