Configuration Management in Software Development Life Cycle

Tejinder Kaur
StudentComputer Application Department, GZS PTU Campus, Bathinda.
Tejkuri@gmail.com

Sanjay Bhatnagar
Associate Prof. Computer Applications Department, GZS PTU Campus, Bathinda
bhatnagar.sanjay@hotmail.com

Deepali
Assistant Prof. Computer Application Department, GZS PTU Campus, Bathinda
Singladeepali88@gmail.com

Abstract- *Configuration Management is the discipline of controlling the occurrence of changes in the complex systems. In Software Development Life Cycle, Configuration Management is generally known as Software Configuration Management (SCM).*

In this paper we will discuss about how Software Configuration Management (SCM) is used to plan, organise, control and coordinate the identification, storage and change of software through development, integration and transfer. However, the project is how large or small Software configuration management (SCM) has a critical effect on quality of the project. SCM uses some Elements for efficient development and maintenance, so that the integrity of the software will never be compromised. It also uses SCM Toolsets for configuration management. Basically, this paper shows that how efficiently SCM works and tackles the changes in the software development.

Keywords-*Software Configuration Management (SCM), Configuration item (CI), Baseline, Software configuration toolset, and Configuration Control Board (CCB).*

I . INTRODUCTION

The development of good quality software is a critical element of successful competition for today‟s software market shares. However, software products are becoming larger and more complex; therefore, the development of quality software is neither easy nor rapid. Therefore, various changes occur at different phases of the life cycle of a project. And because changes happen, so developers have to control it effectively. This can be done by software configuration management (SCM). Configuration management (CM) or software configuration management is the discipline for systematically controlling the changes that takes place during development [6, 7, 12]

For SCM, „Bersoff, et al, in 1980 defined a principal “NO matter where you are in the system life cycle, the system will change, and the desire to change it will persist throughout the life cycle.”

The IEEE defines SCM as “the process of identify and defining the items in the system, controlling the change of these items throughout their life cycle, recording and reporting the status of the items and change requests, and verifying the completeness and correctness of the items.”

In brief, SCM is a method of controlling the software development and modifications of software systems and product during their entire life cycle [5], as shown in Fig. 1.1.

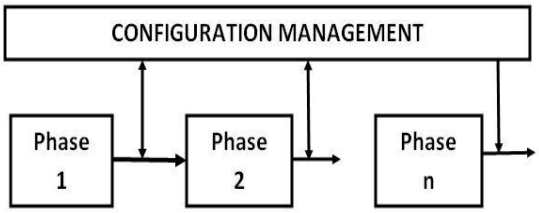

Fig.1.1 Configuration management and Development proces.

II. IMPORTANCE OF SCM

“There is nothing permanent except change.”
Heraclitus 500 B.C. Change may occur at any stage/phase of the development of the software. These changes may be in any area of the software product, i.e. it may be in business requirements, technical requirements, user requirements or any other type of changes may be there. Therefore, it is very important to control these changes. If you don‟t control these change, then it controls you. And that‟s never good. It‟s very easy for a stream of uncontrolled changes to

turn a well-run software project into chaos. For that reason, SCM should be an essential part of a good project management. SCM affects a product‟s whole life-cycle by identifying software items to be developed, avoiding the unnecessarily confusion when changes to software product occur.

The need for CM further arises as software can exist in two different forms− executable and non-executable. The executable form consists of the sequence of instructions that can be executed on the computer hardware. Where, non-executable from comprises documentation such as the requirements specifications, design specifications, code listings, and various documents describing the software. Since, in the early stage of development, software may not exist in executable form, so changes can be easily controlled but gets complicated towards the end of the developments.

The primary goal of SCM is to improve the control management with which changes can be accommodated and reduce the amount of effort expended when changes must be made.

III .ELEMENTS OF SCM

Whenever one discusses about SCM, it introduces a set of questions that are:

- What is system configuration and what are the configuration items?
- How should changes to the configuration be controlled?
- What changes have been made n the configuration?
- Is the system being built to satisfy the needs?

All these questions lead us to the definition of four major elements of the SCM – Configuration Identification, Control, Status Accounting and Auditing. These elements are shown in fig 2.1.

Configuration Identification:

The basic goal is to manage the configuration of the software as it evolves during development. According to Leon [13], configuration identification is a process where a system is divided into uniquely identifiable components for the purpose of software configuration management. These components are

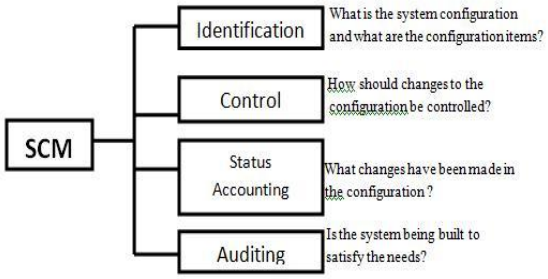

Fig.2.1 Elements of SCM

known as *Configuration Items (CI).* A CI can be a unit or a collection of lower level items [16]. IEEE (IEEE Std. 610.12-1990)[10] defines configuration identification as an element of SCM, consisting of selecting the CIs and recording their functional and physical characteristics in technical documentation. Each CI must be named and versioned uniquely to distinguish it from the other CIs and from other versions of CIs.

In the configuration identification phase, a project's baselines and their contents are also identified. *A baseline* is a software configuration management concept that helps us to control change [13]. It is a document or product that has been formally reviewed and that thereafter serves as a basis for further development. It can also be an assembly of CIs, an accepted configuration [17]. The most common baselines of software development are shown in Fig 2.2 below [15]. The life cycle model shown is a traditional waterfall model; every phase produces a baseline.

The IEEE (IEEE Std.No.610.12-1990)[10] 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 that can be changed only through formal change control procedures.”

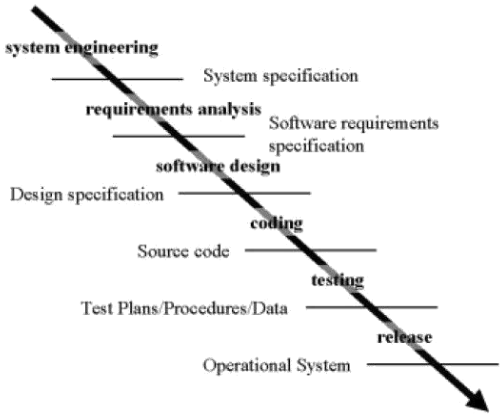

Fig. 2.2 The most common baselines of a software.

At the time when a Baseline is established represents the software in most recent state. After changes are made to this baseline, the state of the software is defined by the most recent baseline and the changes that were made. Among common baselines, *Functional baseline* is generally the requirements document that specifies the functional requirements for the software. *Design baseline consists* of the different components in the software and their designs. *Product baseline* represents the developed system.

The first baseline (i.e. the functional baseline) may consist of only one CI- the requirement document. As the development precedes the number of CIs increases and the baselines gets more complex. New baselines are generated as other software components are integrated. The baselines provide a development and test environment for the new software components undergoing development and integration.

Every CI should have a unique control authority that decides what changes will have to be made in a CI. The control authority may be an individual or a group of people. Three levels of control authority are normally distinguished in a software project:

- software author
- software project manager
- review board

Software Author: Software authors create low-level CIs, such as documents and code. Document writers are usually the control authority for draft documents. Programmers are normally the control authority for code until unit testing is complete.

Software project manager: Software project managers are responsible for the assembly of high-level CIs (i.e. baselines and releases) from the low level CIs provided by software authors. Software project managers are the control authorities for all these CIs.

Review board: The review board approves baselines and changes to them. The review board is the control authority for all baselines that have been approved or are under review. The review board may decide that a baseline should be changed, and authorises the development or maintenance team to implement the change. Control authority rights are not returned. Review boards should be composed of members with sufficient authority and expertise to resolve any problems with the software. The nature of the review board depends upon the project. On a small project it might just consist of two people, such as the project manager and a user representative. On a large project the membership might include the project manager, the technical manager, the software librarian, the team leaders, user representatives and quality assurance personnel.

Configuration Control:

Changes are normal in the evolution of a system. Uncontrolled changes can lead to chaos. Proper change control is the important part of good CM. Therefore, after the configuration items (CI) of the system have been identified, the next step is to control the changes to the software. Configuration control focuses on managing changes to the different forms of the CIs.

The reality of configuration control in a modern software engineering context has been summed up beautifully by James Bach [BAC98]:

"Change control is vital. But the forces that make it necessary also make it

annoying. We worry about change because a tiny perturbation in the code can create a big failure in the product. But it can also fix a big failure or enable wonderful new capabilities. We worry about change because a single rogue developer could sink the project; yet brilliant ideas originate in the minds of those rogues, and a burdensome change control process could effectively discourage them from doing creative work."

Bach recognizes that we face a balancing act. Too much change control and we create problems. Too little, and we create other problems.

According to (IEEE Std. 610.12-1990), baselines can be changed only through formal change control procedures. That is shown below in the figure 3.3.

The *engineering change proposal* is the basic document that is used for defining and requesting a change to a CI. This change proposal describes the proposed change, reasons for it, baselines and CIs that are affected, and cost and schedule impacts.

The engineering change proposals are sent to a *configuration control board (CCB).* The CCB is a group of people responsible for configuration management. The CCB evaluates the change request based on its effect on the project, and the benefit due to change. Considering this analysis as a base, the CCB may approve or disapprove the changes. If a change is approved then it is the duty of the CCB to inform all the parties affected by the change. Traditional procedure of SCM is shown in Fig 2.3.

One important factor in configuration control is the procedure for controlling the changes. Once an engineering change proposal has been approved by the CCB, the actual change in the CI will occur. The procedures for making these changes must be specified. One method for controlling the changes during the coding stages is using program support libraries. A program support library is a repository of authorities and approved versions of the different software components.

The change process is a miniature software development itself. Problems

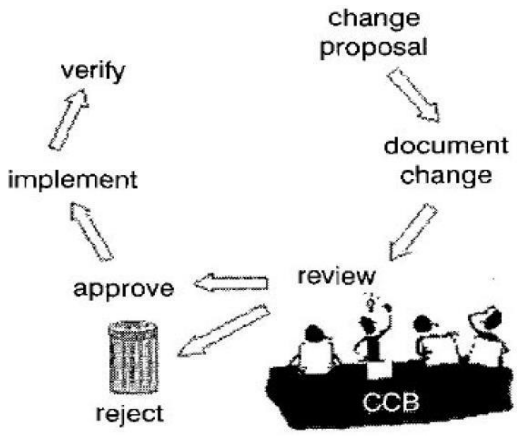

Fig.2.3 Change control procedures in conventional SCM.

must be identified, analysed and a solution designed and implemented.

Status Accounting:

Software configuration status accounting is

„the recording and reporting of information needed to manage a configuration effectively, including a listing of the approved configuration identification, the status of proposed changes to the configuration, and the implementation status of the proposed changes. The status of all configuration items must be recorded. Configuration status accounting continues throughout the life cycle.

Incorporating changes, after they have been approved, take time. So, the changes to a baseline occur over a time period. Some mechanisms are needed to a record how the system evolves and what is its current state. This is the task of status accounting.

Its basic function is to record the activities related to the other SCM functions. It is largely administrative in nature, but gats quite complex after the product baseline has been established. This is because both the executable and non-executable forms exist at this stage and their correspondence and consistency has to be maintained during changes. Software development produces lots of

information that should be recordable and reportable whenever needed.

Configuration status accounting is an SCM task that answers the following questions:

1) What happened?
2) Who did it?
3) When did it happen?
4) What else will be affected?

Configuration status accounting plays a vital role in the success of a large software development project. When many people are involved, it is likely that

"The left hand not knowing what the right hand is doing" syndrome will occur. Two developers may attempt to modify the same SCI with different and conflicting intents. According to IEEE [9],

"Configuration status accounting consists of the recording and reporting of information needed to manage a configuration effectively, including a listing of the approved configuration identification, the status of proposed changes to the configuration and the implementation status of approved changes."

All this and other information related to CIs and activities concerned with them are thus available for the people involved in the project. Status accounting reports include change logs, progress reports, CI status reports and transaction logs

Auditing:

Configuration auditing is concerned with determining how accurately the current software implements the system defined in the baseline and the requirements document, and with increasing the visibility and traceability of software. Auditing procedures are also responsible for establishing a new baseline.

A software configuration audit complements the formal technical review by assessing a configuration object for characteristics that are generally not considered during review. The audit asks and answers the following questions:

1) Has the change specified in the engineering change proposal been made? Have any additional modifications been incorporated?
2) Has a formal technical review been conducted to assess technical correctness?
3) Has the software process been followed and have software engineering standards been properly applied?
4) Has the change been "highlighted" in the CI? Have the change date and change author been specified? Do the attributes of the configuration item reflect the change?
5) Have SCM procedures for noting the change, recording it, and reporting it been followed?
6) Have all related CIs been properly updated?

According to Leon [13], "The purpose of configuration audits is to ensure that the software product has been built according to specified requirements (Functional Configuration Audit, FCA), to determine whether all the items identified as a part of CI are present in the product baseline (Physical Configuration Audit, PCA), and whether defined SCM activities are being properly applied and controlled".

A representative from management or the customer usually performs such audits. The auditor should have competent knowledge both of SCM activities and of the project [13].

IV. SOFTWARE CONFIGURATION MANAGEMENT TOOLS

SCM tools are software tools that automate and facilitate the application of the SCM best practices. As with a compiler, debugger, and editor, an SCM tool must be an essential part of every software engineer‟s tool kit. It is unrealistic to try to maintain effective SCM without an SCM tool.

Tools that support software configuration management are widely available and their use is strongly recommended. No single tool can be expected to provide all the functionality of the widely available tools, and developers need to define their own software configuration management tool requirements and assemble a „toolset' that meets them. Some are better at change management, whereas others have excellent build management and versioning capabilities.

ANSI/IEEE Std 1042-1987 [9], IEEE Guide to Software Configuration Management, recognises four types of toolset:

- Basic
- Advanced
- Online
- Integrated

This classification is used to organise software configuration management tools. Each toolset includes the capabilities of preceding toolsets.

V.CONCLUSION

Software configuration management is simple in concept but complex in detail. Software configuration management is an umbrella activity that is applied throughout the software process. SCM identifies controls, audits, and reports modifications that invariably occur while software is being developed and after it has been released to a customer. All information produced as part of software engineering becomes part of a software configuration. The configuration is organized in a manner that enables orderly control of change. SCM is a method of bringing control to the software development process and is known as an inseparable part of quality-oriented product development regardless of development method. On the other hand, software configuration management is often considered bureaucratic method, an approach that causes additional work and more documentation.

Software configuration management in simple terms is the mechanisms used to control the evolution of a software project. Software Configuration Management is a discipline of 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.

REFERENCES

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

[2]Pankaj Jalote, IIT Kanpur, "An Integrated Approach To Software Engineering", Second Edition,.

[3]Pankaj Jalote, IIT Kanpur, "An Integrated Approach To Software Engineering", Third Edition.

[4]Crnkovic, I., Pearson Dahlqvist, A., Svensson, "Complex Systems Development Requirements – PDM and SCM Integration". 2001.

[5] Crnkovic, I., Asklund, U., Person Dahlqvist, "Implementing and Integrating Product Data Management and Software Configuration Management. Artech House." A. 2003.

[6]E. H. Bersoff." Elements of software configuration management. IEEE Transactions of Software Engineering", pages 79-87, Jan. 1984.

[7]E. H. Bersoff, V. D. Henderson, and S. G. Siegel. "Software configuration management: A tutorial. *IEEE Computer",* pp 6-14, Jan. 1979.

[8]ESA "Software Engineering Standards, ESA PSS5-0" Issue 2 February 1991.

[9]IEEE 1042-1987. "IEEE Guide to Software Configuration Management. IEEE Standards Collection – Software Engineering, Institute of Electrical and Electronics Engineers", Piscataway, New Jersey.1987

[10]IEEE 610.12-1990. "IEEE Standard Glossary of Software Engineering Terminology. IEEE Standards Collection – Software Engineering, Institute of Electrical and Electronics Engineers", Piscataway, New Jersey. 1990

[11]IEEE 828-1990. "IEEE Standard for Software Configuration Management Plans. IEEE Standards Collection – Software Engineering, Institute of Electrical and Electronics Engineers", Piscataway, New Jersey.1990

[12]IEEE 828-1998. "IEEE Standard for Software Configuration Management Plans. IEEE Standards Collection – Software Engineering, Institute of Electrical and Electronics Engineers", Piscataway, New Jersey. 1998

[13]Leon, "A Guide to Software Configuration Management.", Boston: Artech House. A. 2000.

[14]Watts S. Humphrey, "Managing the Software Process, SEI Series in Software Engineering", Addison-Wesley, August 1990.

[15]Pressman," Software Engineering: a Practitioner's approach, 4th ". New York: McGraw-Hill Companies. 1997 [16]Rahikkala, "Towards Virtual Software Configuration Management. A Case Study. Espoo, VTT Electronics", 110 p. + app. 57 p. VTT Publications 409. 2000.

[17]Taramaaa, "Practical Development of Software Configuration Management for Embbed System".1998.

[18]"The STARTs Guide - a guide to methods and software tools for the construction of large real-time systems", NCCPublications, 1987.

[19]Walter F. Tichy: "Tools for Software Configuration Management", In *Proceedings of the International Workshop on Software* Version *and Configuration Control*, Grassau, Germany Jan 27-29, 1988