A Change Process Model in an SCM Tool

Ivica Crnkovic
ABB Industrial Products, 721 67 Västerås, Sweden
ivica@sw.seisy.abb.se

Abstract

This paper is a survey of a change process model developed and used at ABB Industrial Products. The change process is based on a Software Configuration Management (SCM) tool and the Capability Maturity Model (CMM). CMM emphasizes the importance of controlling the changes made in a development project. The SCM tool, developed at ABB Industrial Products, is a change oriented tool by means of which changes are managed through Change Requests. A Change request is a document which describes changes processed in a software. During the development process, Change Requests collect information about the changes implemented in configuration items and they pass through different states. Since Change Requests are under version control, the data obtained from them give information about both the number and types of changes and the change process behavior. Experience has shown which an SCM tool that supports Change Management is very important in large software systems, especially in the project planning, verification and maintenance phases.

1. Introduction

ABB Industrial Products develops different families of process-control systems. Several hundred programmers are involved in this development. The quality and functionality of the software configuration management must be of a high order. It must be possible to reproduce earlier software releases, correct any errors in these, and if required implement the same changes in other products, or product versions. For these reasons SCM tools have been essential parts of development work at ABB Industrial Products for more than 10 years.

The company has recently reached the CMM level 2 [1]. Since the SCM process is a key process area at the CMM level 2, SCM questions have been focused on the process rather than on the tools. During the period of CMM introduction at the company, the main issue was how to implement the SCM process and relate its activities to the existing SCM tool and methods. The change process was a particular focus of interest. Since the SCM tool used in the company is change-oriented, the features from the tool are utilized in the SCM process definition.

Change-oriented SCM tools deal with logical changes introduced into software, rather than with component versions, the approach of Version-oriented tools [2]. A change management process defines different roles for different groups of people involved in a software development process. The people responsible for the product (managers, project leaders) are interested in changes on the general functional level. Quality assurance personnel are interested in the changes made which necessitate testing, verification and documentation of the relevant parts of the software. Change-oriented SCM tools give better support to the implementation of SCM processes as specified in CMM.

This paper describes how a Change Management process is defined in relation an SCM tool which supports Change Management.

Chapter 2 gives an overview of SDE, the SCM tool. The basic characteristics of Version Management and Change Management are described here.

Chapter 3 describes the Change process in a development project. The process is designed according to CMM requirements for SCM planning, verification and measurement.

The last chapter contains examples of measurements performed on data collected by the SCM tool in a development project. These measurements contribute to an understanding of the development process and make possible improvements the process.

2. Overview of SDE - a Change-oriented CM Tool

SDE (Software Development Environment) is an SCM tool based partly on Revision Control System (RCS) [3]. Using slightly modified RCS commands and certain new commands related to RCS files, SDE enables easy and fast browsing through hierarchical system structures and versioned files. The SDE and RCS commands are encapsulated in GUI-applications.

SDE includes the basic features of an SCM tool: Version Management, Configuration and Build Management,

Change Management, Work Space Management and Product Management.

This chapter briefly describes version and configuration management, and change management implemented in the tool.

2.1 Version and Configuration Management

Version management embraces identification all of the items of a software system. The RCS package is used for version management at the file level. Versioned files are deposited in RCS libraries. To modify a file, a developer must check out a specific file version from an RCS library. The modified file is then checked back in, and a new file version created in the RCS library.

In addition to managing file versions, SDE operates on the versions of structures, designated SDE software systems. An SDE system contains system versions, called system configurations. A configuration consists of a tree structure of subsystems. A subsystem collects files which make a logical (sub)function of the system. The files are under version control. The existence of different configurations makes possible parallel development and maintenance of different product versions.

The *baseline* process in a configuration consists of the tagging the selected file versions with a baseline name. The versions specified in this way are used for the building of a final version of a software product. Several baselines can be included in a system configuration and eventually an approved baseline will be used for a new product version.

The file version selection depends on the type of the development process. In a rapid development model, typically, the latest versions of files will be selected. In a more formal approach the file versions will be selected on the bases of well defined criteria and will pass through a quality assurance process, such as code review and testing. In such a case the versions are selected in accordance with a version *State*. SDE provides support in the management of state of file versions.

Figure 1 shows an example of a process of changing states of file versions.

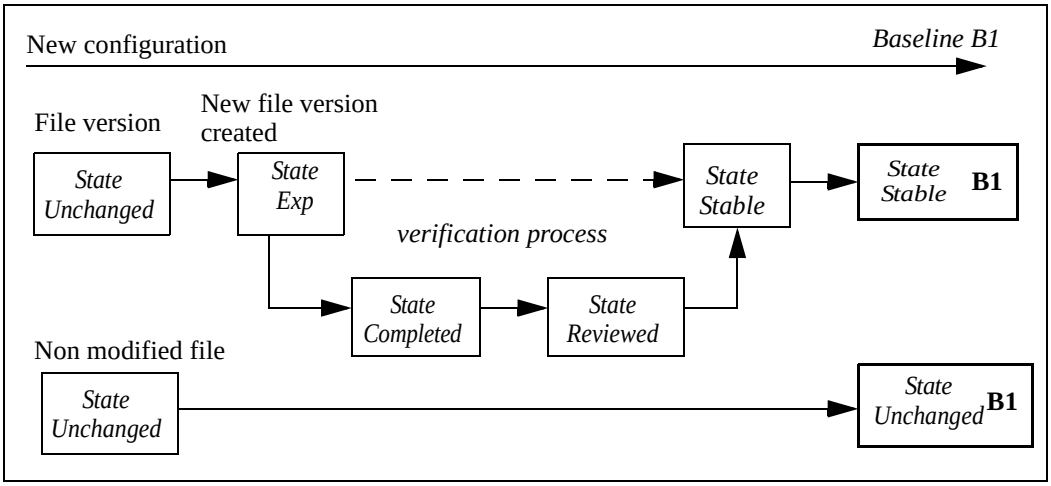

FIGURE 1. Changes in version states in the development process

A new development or maintenance begins with the creation of a new system configuration. The file versions imported from the source configuration are given the *Unchanged* state. If a developer modifies a file, the new file version automatically enters the *Exp (Experimental)* state. The following state management depends on the verification process. Typically, when a developer completes the changes, he/she sets the *Completed* state on the file version. The changes performed can go through the code-review process and eventually the file ends in the *Stable* state. In the baseline procedure, the stable file versions are tagged by a baseline name.

Additional selection requirements can be defined through the change process: Only those file versions which are parts of approved changes are selected for the new baseline.

2.2 Change Management in SDE

Any change performed in SDE is under change-set control. A basic item of SDE change management is a Change Request (CR), an entity which describes a logical change to be performed in a software system. Change Requests are created from Requirement Specifications or from error reports. During the development process CRs collect infor-

mation about physical changes made in the system: When a developer checks in a modified or a new file, he/she refers to a related CR. The file name and version are registered in the CR. The final version of a CR includes both a description of a logical change made and information about all modified file versions.

Every SDE system configuration includes a CR library - and therefore all the logical changes introduced in a software version are contained in this.

Change Requests are implemented as text files which follow a specific syntax. The header part of a CR includes keywords such as Priority and CR Type, creation date and termination date. A list of files being checked in follows. The body part includes a description of the change and log messages of the files checked in. Figure 2 shows an example of a CR:

```
CR-History-Parameters
Term  icrnkovi          1997/12/03 13:17:32
Change input parameters for History command

Terminated:  97-12-03
Created:     97-01-13 by icrnkovi

Function:  rcshist.exe
Reason:    Improvement
Priority:  Medium

File:      ./history/rcshist.vbp 1.2
File:      ./history/frmdummy.frx 1.1
File:      ./util/sde/classhistory.cls 1.4

Description:
Modify input parameters so that they work
for File Manager, Explorer, Visual studio
and Visual Basic.

Log Messages:
./history/rcshist.vbp 1.2
  Initial Version
./history/frmdummy.frx 1.1
  Initial Version
./util/sde/classhistory.cls 1.4
  Add ShowHistoryLine method in the SdeAc-
  tiveXVB control
```

FIGURE 2. A Change Request Example

CRs are modified by different commands, not manually. For example, the original RCS **ci** command has been modified: When a developer checks in a file he/she selects a CR. The selected CR will be checked out, modified and checked in. In this way a CR collects information about modified file versions. A **check in** dialog box that uses the **ci** command is shown in Figure 3.

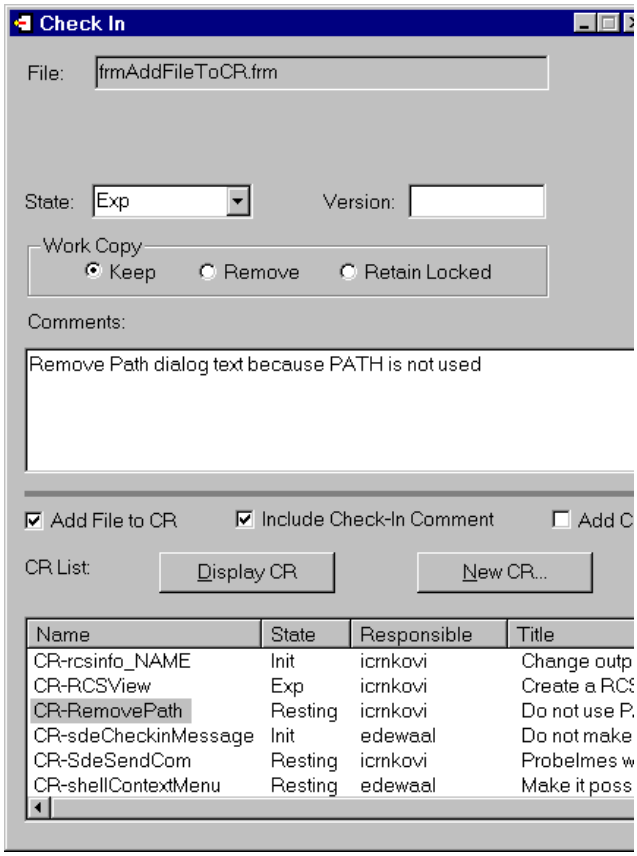

FIGURE 3. Check in a file and select a CR

Change Requests are under version control. Each time a CR is changed it is automatically checked out and after the modification checked in. CRs are saved in a RCS directory located in a CR library. As a versioned file under RCS control, a CR also includes attributes from the RCS: a state, a responsible user (author) date of change and other RCS attributes. A history of a CR is shown in Figure 4.

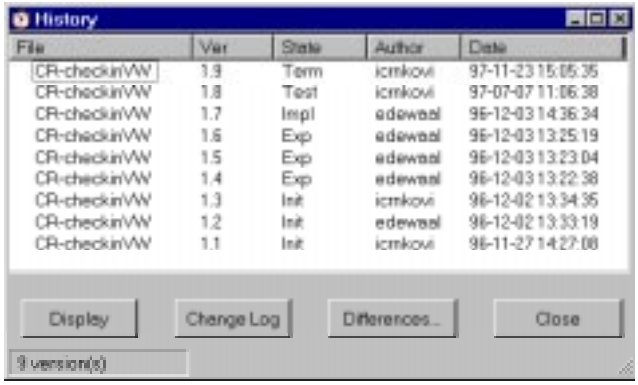

FIGURE 4. Change Request History

3. Change Management Process

CMM requires that CM activities are planned and that the changes introduced in the software are under control. Software Configuration Control Board (SCCB) approves the changes entered in the new software baseline. Change Requests are used in SDE for planning, following up and approving changes entered in a software version. SCCB and the SCM group (a group with special access rights to the SCM structures) define Change Requests to be implemented in the project.

Change Requests originate from development plans, identified as Development items, and from error reports. When created, Change Requests are updated by the developers. When a developer checks in a modified or a new file, he/she refers to a related CR. The file name and version are registered in CRs. In the verification phase of the project SCCB approves completed CRs which are used as a basis for creating a baseline (Figure 5).

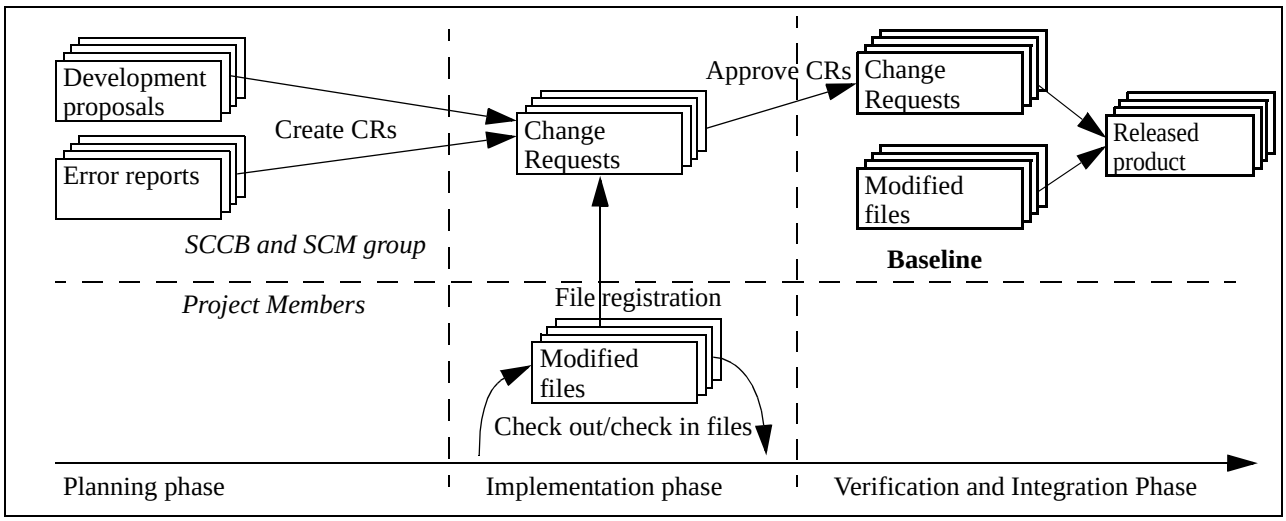

FIGURE 5. Controlling Change Requests during the development process

After each baseline new Change Requests can be created. They define new activities or possible corrections of the previous baseline.

In a maintenance project Change Requests originate from error reports. One error report can produce several Change Requests if different parts of software have to be changed.

In the final stage of a project a report listing CRs is produced. The listing can be used as a basis for testing the new product version.

A Change Request passes through different states during the development process. When a CR is created it is in the *Init* state. During the work sessions it passes through other states, such as *Exp (Experimental)*, *Implemented*, *Tested* and eventually reaches the *Terminated* state. The CRs integrated in a product release are in the state *Released*.

Figure 6 shows the different states through which a CR passes in a development process.

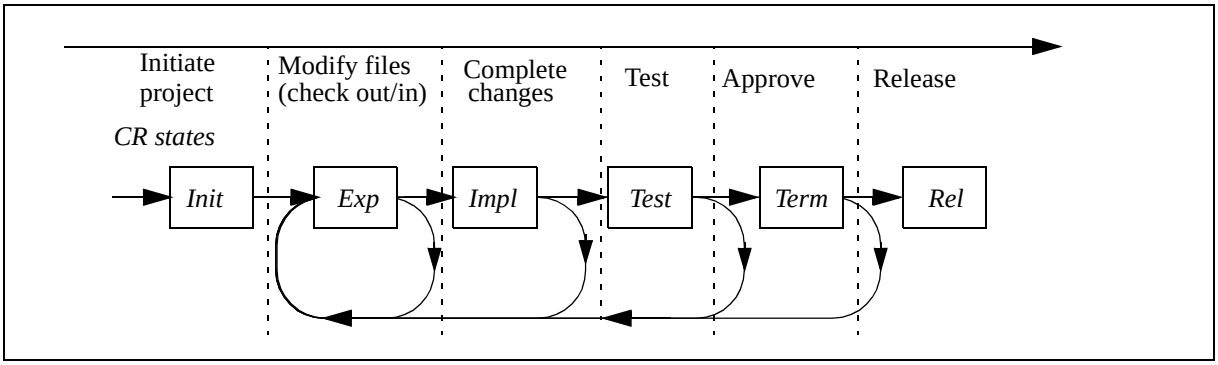

FIGURE 6. Change Requests in a development process

4. Change measurements

The information stored in Change Requests includes much data which can be used for software metrics. The number of changes, their states, classification according to priority, type or function, number of changed files, etc. is a basis for size-type metrics. Since CRs are under version control, the history of every change is also available. The states of all changes are available for the whole period of the development process. The data from the state change in a time period is input to another type of metrics - process metrics [4].

A CR-Metrics application, included in SDE, collects data from CR libraries, by parsing all versions of all, or specific, CRs. Information about every CR version, such as creation date, state and author are taken. The results of measurement are displayed as embedded Excel objects [5] in form of different graphs.

All measurements are performed in a similar way. All versions of all CRs are parsed and different criteria for extracting data are used. Examples of the type measurements are listed below:

- **CR Current status** -The current states of CRs for each period are extracted. The result is presented in a time graph which shows the dynamic of different CR activities.
- **Accumulated CRs** - CR versions are sorted according to date and classified in two groups: completed, and not-completed. The result is presented in a time graph.
- **Latest Changes** -The CRs which have changed a state for a given period are extracted.
- **CR life length** - A distribution of CR life lengths is shown. The life length of a CR is the time between the creation and the final modification date.
- **CR Type and Priority** -The latest versions of CRs are classified according to CR type (Error, Improvement, New Function) and priorities are shown.

Figure 7 shows an example of a current status graph. The graph shows the number of CRs created during a development project's life. CRs are classified according to their states: *New* - those which are created but no work has started on them, *Open* - those which are in the process, *Resting* - those for which SCCB has decided not to process in the current phase of the project, and finally *Completed* - those which are implemented and finished.

The graph shows accumulated values, i.e. the history of the process can be seen from it.

.

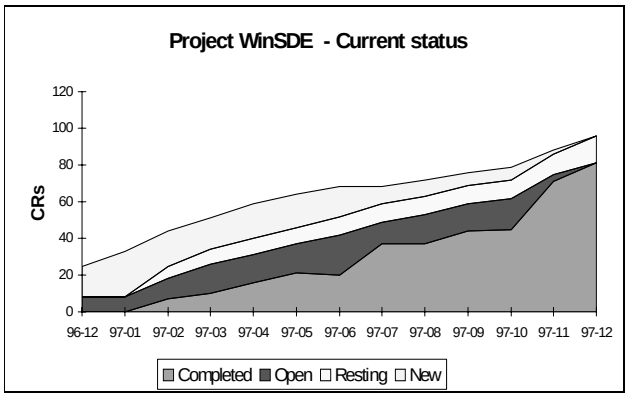

FIGURE 7. Current States of CRs during the development process

This project has used a development model which is a combination of the spiral model and the evolutionary prototyping model [6]. The development was performed in a number of iterations, each iteration consisting of several phases: prototyping, evaluation of different alternatives, refining the prototype, developing and building the deliverables and a plan for the next iteration. An implication of this model is the constant increase in the number of CRs - new requirements are defined during each iteration (refinements) and at the beginning of the new iteration (new functions). The number of new CRs increases constantly. The number of completed CRs also increases, but some irregularities can be seen - when the time for a new deliverable is approaching, a many CRs are terminated.

Figure 8 shows the same type of graph for another project which followed the Waterfall model. In the project initiation phase, all the requirements have been defined and CRs have been created. The graph shows how the work has improved. The number of open CRs increases, and in the final, test phase, the CRs are being completed.

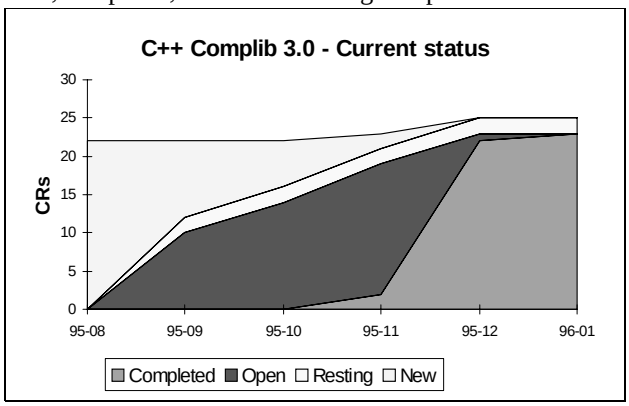

FIGURE 8. Current states of CRs for a project using the Waterfall model

The objective is to use CR-metrics during the project when the project manager, SCCB and project members can follow up the project status. The final measurements can be performed after completion of the project. The metrics can be compared with those from other projects and related to other facts.

5. Conclusion

The paper describes an SCM process model based on SDE, a change-oriented SCM tool. The model has been recently applied in new development projects. The use of the tool has not been dramatically changed, but the focus has been moved from the tool to the process. While version management has remained as before, change management has become more important.

A new project is started with the registration of new functions which are intended to be developed. When a function is implemented the corresponding CR is closed. The baseline process starts with a decision regarding which closed CRs will be included in the baseline. The corresponding functions are then frozen. In a later development phase, new CRs are introduced when the frozen functions need to be changed (improved, or corrected). These new CRs describe in more detail the changes which are to be made. When a project approaches the final state, the treatment of CRs is more formal. For example, after a beta release, SCCB decides for each individual CR if it should be processed or left for a next phase.

Focusing on change management has increased a possibility of supervising a project and it has lead to better planning under the project work. The awareness of the project state increases as the project advances to its final stage.

Experience has also shown that adequate support for the CM process is indispensable. The SDE concept based on Change Requests facilitates the process implementation.

References

1. Mark C Paul, Bill Cutris et al, *Capability Maturity Model for Software, Version1.1*, Technical report CMU/SEI-93-TR-024, Software Engineering Institute, Carnegie Mellon University
2. Ian Sommerville (Ed.), Software Configuration Management - Introduction, *Software Configuration Management ICSE'96 Workshop, Berlin, Match 1996, Selected Papers, Springer Verlag, ISB N 3-540-61964-X*, pages 1-7
3. Walter F. Tichy, RCS - A System for Version Control, S*oftware and Practice Experience*, 15(7):635-654, 1985
4. The Software Measurement Laboratory, University of Magdeburg, *http://irb.cs.uni-magdeburg.de/sw-eng/us/metclas/index.shtml*
5. Microsoft Visual Basic 5.0 ActiveX Controls Reference, 1997, ISBN 1-57231-508-3
6. Steve McConnell, *Rapid Development: timing wild software schedules*, Microsoft Press, 1996, ISBN 1-55615-900-5