Experience with Change-oriented SCM Tools

Ivica Crnkovic
ABB Industrial Systems AB, 721 67 Västerås, Sweden
E-mail: ivica@sw.seisy.abb.se

Abstract. In the recent years two approaches in Software Configuration Management (SCM) tools have been emphasized: a Change-oriented model and a Version-model approach. This paper gives an overview of two Change-oriented SCM tools developed at ABB Industrial Systems and describes the experience with their usage. The first tool is strictly change-oriented and it requires formal consistency of the entire software system. The second one is more pragmatic and less formal. It uses both Version and Change approach. The experience shows that a tool that supports a Change Management is very important in large software systems, especially in the verification and maintenance phase. Change Management is not only used as a part of a SCM tool, but in the entire development/maintenance process; in planning, producing release documentation, etc. However, a usage of a strictly change-oriented tool has shown that programmers find the method too complicated and unpredictable. The second tool that controls software component versions, but also supports Change Management in the development process appears to be more effective.

1 Introduction

One of the topics discussed on SCM-6 Workshop was the importance of having a change-oriented approach in a Software Configuration Management process [1]. Change-oriented SCM tools deal with logical changes introduced in software, rather than with component versions which is the approach for Version-oriented tools [2].

A change management process defines different roles for different group of people involved in a software development process. The product responsible people (managers, project leaders) are interested in changes on the general functional level. A change view in a form of parts of software that should be tested, verified and documented is of interest of quality assurance people. The developers which implement changes are interesting in a version-file view. They want to work with particular file versions.

This paper describes experience with using two different SCM tools at ABB Industrial Systems, both developed internally. The first tool, called **MaMethod** is strictly change-oriented, while the second tool, called **SDE**, represents a combination of version-oriented and change-oriented methods.

The section 2 shortly describes the MaMethod tool and the experience of using change sets. The section 3 describes the usage of change sets in **SDE**. The change sets are used not only for the software configuration management, but also in other parts of the development process. A final conclusion is given in section 4.

1.1 Background

ABB Industrial Systems develops several families of process-control systems. The final products are combinations of both developed and bought software and hardware. Several hundreds of programmers are involved in the development. Programmers are divided in development groups, where each group is responsible for some components or for their integrations in the final products. Both components and products are delivered in a form of releases - a specific product version includes well defined component versions.

Very high requirements are put on the products - they must run round the clock 365 days per year. The products are supported 10 years. Several versions and variants of the same products are maintained.

These conditions put high requirements on the software quality and especially on software configuration management. It must be possible to re-produce an old software release, correct errors in it, and possibly implement the same changes in other products, or product versions. For theses reasons ABB Industrial Systems has used SCM tools as essential part in the development process in more than 10 years.

2 Using a Change Oriented CM Tool

2.1 MaMethod Tool

MaMethod is a SCM tool developed in middle of eighties as a part of the entire development environment [3]. At that time all the software was developed internally, and the same was true for MaMethod.

MaMethod includes basic features of an SCM tool:

Version Management, Configuration and Build Management, Change Management, Work Space Management and Product Management.

- ***Version Management*** comprehends identification of versions of software modules (files). The files are not treated individually, but as parts of a hierarchical structure called software systems. Both source files and files generated by build procedures are defined as parts of systems. A file version, independently if a source or a generated file, is identified by its “logical” name and a version number. Every file version is implemented as a file. When working in the MaMethod environment, the programmers use logical names, and MaMethod takes care to bring a proper file version.

  A file version that belongs to a system version is not necessary placed physically in the system structure - typically, when several system versions contain the same file version, they point to the same file.

- ***Configuration Management*** includes procedures for comprising selected file versions in a set. MaMethod starts the configuration from the system level. A specific system version is a configuration of identified subsystems versions, where a subsystem version consists of particular file versions. A subsystem version also include references to other subsystem versions which include files used in the build procedure. In that way a subsystem version defines a complete environment required for its modification and building.

  MaMethod users work always in a specified configuration and automatically get the belonging file versions.

  The MaMethod build facility is similar to make. A build procedure called EXECUTE parses trough build files to find dependencies between files, finds the appropriate file versions, compares their dates, checks their status and if necessary executes the build procedures described in the build file. The build procedures treat input (source) and output (generated) files in a more formal way than Make - they are specified as variables and build procedures are defined as functions of these variables.

  A creation of a new system configuration is strictly related to change sets and it will be presented later in more details.

- ***Work Space Management*** is tightly connected to Configuration Management. MaMethod users login into a specific system configuration. The entire configuration space consists of three levels: A reference level where frozen baselines of a system are stored, a work level that collects changed files and working configurations, and finally private working structures where files are actually changed.

  In order to change a file, a user has to *borrow* it. The file is locked on the common work level and copied to the user's private workspace. The user modifies and possibly builds and tests the changes locally in his/her private workspace. The new file version is available for other users when the user *returns* the file.

  The work level is also used for generation of new system versions. Several working versions can be created on the work level. The working system versions refer to files placed in the work and reference structure. When users build the system, or a part of it, in the private working structure, they refer to files they have modified from their structure, and other files from a common system working version. One of system working versions can eventually be taken as a new official version and copied to the reference level (Figure 1).

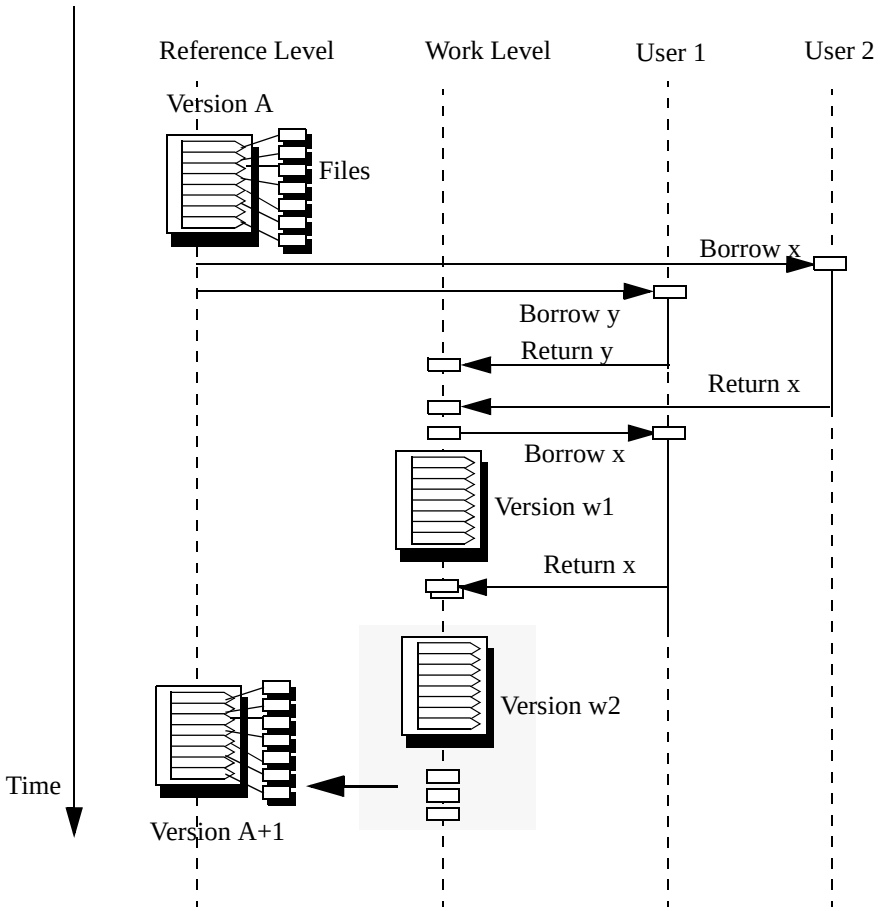

Fig. 1. MaMethod Work Space

2.2 Change Management

2.2.1 Change Registration

Any change done in MaMethod is under a change-set control. A logical change item is defined as a *Change Request (CR)*, an entity that describes a change to be done in the system. When a user borrows a file, he/she must to refer to a CR. When a file is returned from the user working space, the filename version is registered in the related CR (or CRs). In this way a CR collects all the filenames and their versions being changed.

2.2.2 System Integration

A new system version created on the work level consists of file versions from the original system version and from the selected modified files. The modified files are not selected directly, but Change Requests which are planned to be included in the new system version, are selected. The file versions that belong to the selected CRs are identified by MaMethod and integrated in the new system version (Figure 2).

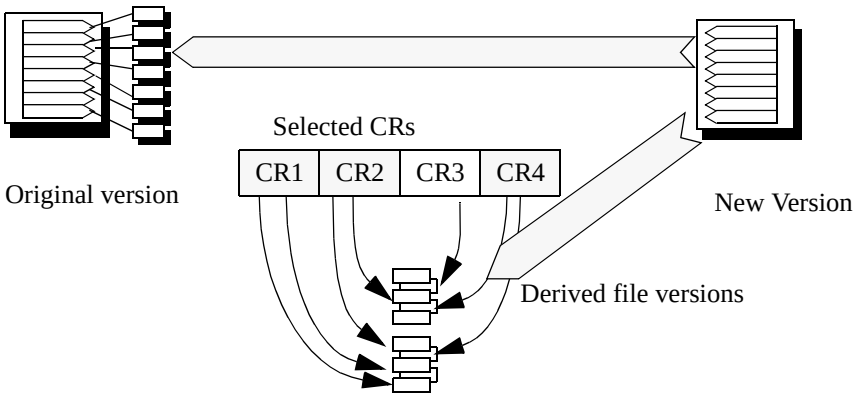

Fig. 2. New system version based on the original version and selected CRs

The advantage of this approach is that planned logical changes are directly included into a new software version, and the mapping between logical changes and physical files are done by a tool, not by programmers.

2.2.3 A Consistency of a System Version

There exists cases when file versions pointed from the selected Change Requests can not directly be included in the new system version, because the generated system would not be uniquely defined.

For example, several CRs can refer to the same file but different versions. This situation occurs when programmers change one file due to several Change Requests. In that case the work space will contain several versions of one file. MaMethod does not build a fictive file version that includes changes related to the selected CRs. Instead of that, the latest version of those selected is taken. It is obvious that the latest version includes some changes that belong to other CRs. By simply taking the latest version of the file, a code that belong to another change will be also included into new system version. If this is only a part of another Change Requests, i.e. if some other files have been modified due to the same Change Request, than an inconsistent combination of files will be generated.

Different Version-oriented SCM tools treat this problem in different way. While some of them ignore the problem, others give a warning for an inconsistent system [2].

MaMethod finds always a minimal set of CRs that make a consistent set of file versions:

1. If several CRs include the same file but different versions, the latest version of them will be taken.
2. For every CR included in the integration, also their **connected CRs** are taken.

   The connected CRs are defined in the following way:

   We define a CR *A* as a set of file versions:

   $A = \{f_1.v_1, f_2.v_2, ..., f_k.v_k\}$, where v_i denotes a version of a file f_i included in a CR *A*.

   Suppose we have two CRs: *M* and *N*.

   If there is a file f_i, where file version $f_i.v_m \in M$, and $f_i.v_n \in N$, than CR *N* **is connected to** CR *M* if $n \leq m$.

The following example illustrates the rule:

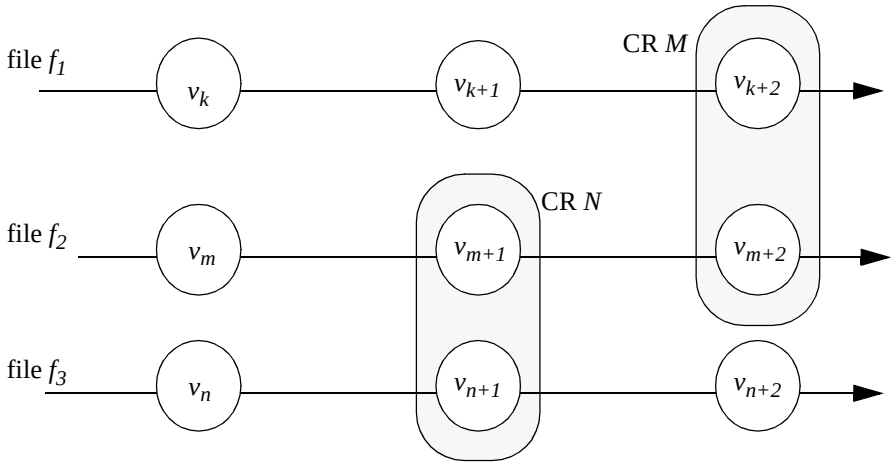

Fig. 3. Connected CRs

CR *N* is connected to CR *M* and it will be taken if CR *M* is selected. File versions $f_1.v_{k+2}$, $f_2.v_{m+2}$ and $f_3.v_{n+1}$ will be integrated in the new system version.

If CR *A* is connected to CR *B* and CR *B* is connected to CR *C*, then CR *A* is connected to CR *C*. This implies that a selection of connected CRs is a recursive process.

Including specified and all connected CRs into a new system version, MaMethod ensures that the new version is consistent, but it also can lead to unpredictable results - some unwanted changes can be included into the new version!

2.2.4 Experience of using Change Sets

MaMethod have been used almost ten years. The experience of using Change Sets is both positive and negative.

The positive experience in using a Change-oriented tool is first of all in having a better control of delivered product versions. Each product version is followed by an automatically created document that lists all functional changes included in the release. The generated list of changes is also used for tests and verifications - those changes that are implemented in a new version are verified in the final product test. It is also very convenient to find what changes have been made in an old product version and which files have been changed due to a functional change.

The negative experience comes from both principal problems and programmers way of working.

Programmers are focused on changing of code. It is very tempting to add some new “small” changes in already borrowed file without registering the new change. The implication of this is that a released software actually include more changes than it is specified in Change Requests. Formally, you are never sure if the documentation really describes all changes introduced in the release. Hopefully the biggest and most important changes are registered.

Another problem is in the treatment of the connected CRs. Some CRs have to many connected CRs. Such a situation happens if there is a “common” file that is related to a lot of other files in a system, for example a build file or a common include file. Such a file can include changes that belong to several CRs. In that case the connection between CRs depends on the order in which the CRs have been made. This is an irritating moment for programmers. To avoid the overall dependencies, the programmers tend to isolate the changes in the common files by creating special CRs that define changes only for these files. This problem could be solved if each file is created dynamically by merging code parts from the change sets. Unfortunately this process is not possible to run automatically, and manual merging of files in a big software system is out of the question - it takes too much time and it is too risky!

Using change sets works very good in the maintenance process, where not too much changes are introduced, and when it is important *what* changes have been made. In the development process, change sets become of less importance.

In general the following conclusion can be done:

While quality people and managers find the change-oriented approach convenient because of possibility of tracking the changes, the programmers feel that the methods are too formal and inflexible.

3 Using a Combination of Change and Version Model

In the beginning of nineties our organization started the development of a new generation of products, based on open architectures. The main goal of new development process was efficiency and quality: Programmers should concentrate on pure development issues, other activities should be managed by tools as much as possible. Standard software available on the market should be used as much as possible for both development tools and product components. Development tools should be integrated in a common graphical user environment, and data between different tools should be exchanged as much as possible.

The new development process has required a new, modern and efficient SCM tool. The new SCM tool was developed internally[1], this time based on Revision Control System (RCS) [4]. The new tool is called **SDE** (Software Development Environment) [5].

3.1 SDE Basic Characteristics

SDE is a software package adjusted for the development of large systems, to some extent similar to MaMethod. It defines structures of repositories where versioned files are saved. The repositories, called SDE software systems, are divided into configurations which represent different versions or variants of software systems. Each configuration is a hierarchical structure consists of subsystems, where each subsystem contains files under version control. The separation in configurations makes it possible to develop different versions of software in parallel.

SDE Workspace for programmers is placed in SDE projects, where each project member has a private working structure. Such working structures are identical to system structure. While SDE software systems include all versions of files, SDE projects have a view to specific set of versions of files. Project members get their working versions by checking out files from their view.

1. It may look strange that a SCM tool was developed internally, when the strategy was to buy software on the market. Indeed, the initial intention was to buy a SCM tool. A requirement specification was made and the market has been investigated. A CM tool was selected and prepared for overall usage. Unfortunately, before the tool was established in the organization, it disappeared when the company that had developed the tool, was bought by another one (the new variant of the tool successfully appeared some years later). The second tool that was selected, showed too poor quality when used in a large, complex environment.

   The functionality of SDE has grown with time, according to new requirements. While a group of eight people worked on SDE during the first three years, three people work today full time on SDE. The work includes the development, maintenance, porting to another platform and user support. The development does not include only CM tool, but also some others tools related to software development (different utility programs, generation of documentation, etc.) and support for definition internal infrastructure. Several hundreds programmers are using SDE today.

Using slightly modified RCS commands and some new commands related to RCS files, SDE enables easy and fast browsing through the hierarchical system structures and versioned files. On a top of RCS and some SDE commands, GUI-applications are made.

SDE uses Make facility with pre-defined include files that refer to different parts of the entire development environment used for the system building. When programmers log in a project they get all definitions need for make to find proper files, both from the current project and external software systems.

3.2 Using Change Requests

SDE includes support for Change Management. The experience from MaMethod led to a conclusion that Change Management is very useful, but it should not be inflexible. For this reason a combination of a Version-oriented and Change-oriented approach was made in SDE. Change Requests are used to define and to follow up logical changes introduced in a version of a software system.

Every physical change in a software system is related to logical changes, but the integration process i rather related to file versions than change sets.

A CR is implemented as a simple RCS file. A file version is a text file with a specific format. The header part of a CR includes some keywords like Priority and CR Type, creation date and final termination date. When files are modified, and checked in, CRs are updated by their file names and versions. The body part includes a description of the change.

As a versioned file under RCS control, a CR does not include only the change description and list of changed file versions, but also features from RCS: a state, a responsible user (author) and date of change.

CRs are saved in a RCS directory placed in a CR library. Every system configuration includes a CR library - so a CR library comprises all logical changes of a software version.

CRs are not modified manually, but different commands take care on that. For example, the RCS **ci** command has been modified: When a programmer checks in a file he/she also specifies CRs to which the checked-in file version is related, as illustrated in the example bellow:

```
ci -cProblem_010 file
Log message
.
```

The ci command first checks in a file and then checks out CR specified by the -c option from the CR library, modifies it and checks in back. The new CR version includes the file name, version and the version log message of the checked in file.

Figure 4 shows a typical CR:

```
CR-rce_eval Term      lglomsru  1996/03/22 09:26:20   Evaluate RCE
----------------------------------------------------------------

Terminated: 96-03-22 by lglomsru
Created:    96-01-30 by lglomsru
Type:       New Function
Priority:   High

File:       ./doc/rce/rceeval.frm 1.1 1.2 1.3
File:       ./doc/rce/rceback.frm 1.1

Description:
Evaluation report of the application Revision Control Engine - RCE


./doc/rce/rceeval.frm 1.2
   Added intro, gen descr and RCE Functions
./doc/rce/rceeval.frm 1.3
   The revision without comments from others
   Approved by Ivica Crnkovic
```

Fig. 4. Change Request

A more advanced interaction is implemented in GUI applications where available CRs can be selected, displayed, or a new CR can be created during the check in process.

A final version of a CR includes a description of a logical change and information about all modified file versions.

3.3 System Integration

The integration process implemented in SDE is a version-oriented type. There are several possibilities to select files for the new system version - latest versions with a specific state (*Stable*), latest versions created before a specific date, or just the latest versions. In the version-oriented model, programmers have better control over their code, even if a large number of files are processed.

However, the way to processing files goes through Change Requests. Every change done in a system version is initiated by a CR. A programmer works on a change described in a CR. When all changes related to a CR are completed, the programmer sets the CR-state to *Terminated*.

Even if a new software version is generated directly from the specified file versions, there is a possibility to check which Change Requests are included in the new release. SDE has a support to compare the content of a software system with the registered files in CRs. A list showing relations between CRs and files included into the generated system version can be produced.

Figure 6 shows a list of changed file versions and related Change Requests. The *Relation* column shows if some file versions not included in CRs are in the selected file set, or if there is some file versions belonging to a CR that are not included into the file set.

```
File                 Ver  State User   Relation      Change Requests
-------------------------------------------------------------------------

./Encap/Makefile     1.1   Rel   sfrennem  = 1.9:CR-Make  1.11:CR-move
./SDECon/Interf.C    1.11  Exp   sfrennem  > 1.10:CR-SoftB 1.9:CR-new...
./SDECon/sde.pd      1.6   Rel   mmedin    = 1.6:CR-UpdateMakefiles
./Term/Makefile      1.10  Exp   sfrennem  > 1.6:CR-UpdateMakefiles
./cr/Mngr/CRAppl.C   1.27  Exp   edewaal   = 23:CR-SysDir 1.27:CR-crSDE.
./cr/Mngr/CRAppl.H   1.20  Exp   edewaa    < 20:CR-SdeSys 1.21:CR-cr...
./cr/Mngr/CRCont.C   1.3   Exp   icrnkovi  = 1.3:CR-mod   1.2:CR-newCom.
./cr/Mngr/CRCont.H   1.3   Exp   edewaal   = 1.3:CR-mod   1.2:CR-newCom
./cr/pmr/FindCR      1.2   Exp   sfrennem  !
.....................
```

Fig. 5. Comparing integrated file versions with Change Requests

3.4 Experience with Using CRs in a less Formal Way

The main difference in the integration process between MaMethod and SDE is the way of collecting files and generating new system versions. While MaMethod starts from change sets, SDE uses directly file versions for the integration with implicit inclusion of Change Requests. SDE allows, with a warning, that a only part of a Change Request is integrated. The experience is that the pragmatic SDE approach is simpler and easier to control than the more formal approach in MaMethod.

Change Requests are not used only in the software integration process, but they have several purposes:

1. CRs are used as ‘to do’ list. This list is updated from the “Fault Tracking System”, from the “Proposed Development Items” database, or just by programmers who want to record a proposal for a change.
2. CRs are used to follow-up development projects.
3. CRs are used for the release documentation generation

A Change Request passes trough different states during its life. When a CR is created it gets a state *Init*, under work sessions it passes through some states, when eventually ends in the *Terminated* state. Different metrics can be extracted from CRs. One example is a presentation of the current project state - how many CRs exist, how many are completed, how many are currently open. Another example is a presentation and classification of history of CRs - how log time is needed to implement a change, how the change implementation was scheduled.

Figure 6 shows a typical example of a development project life - a project starts with a well defined requirements, but during its life a number of new Change Requests arise.

When the project is closed and the new product version is delivered, there are still some CRs postponed for next releases.

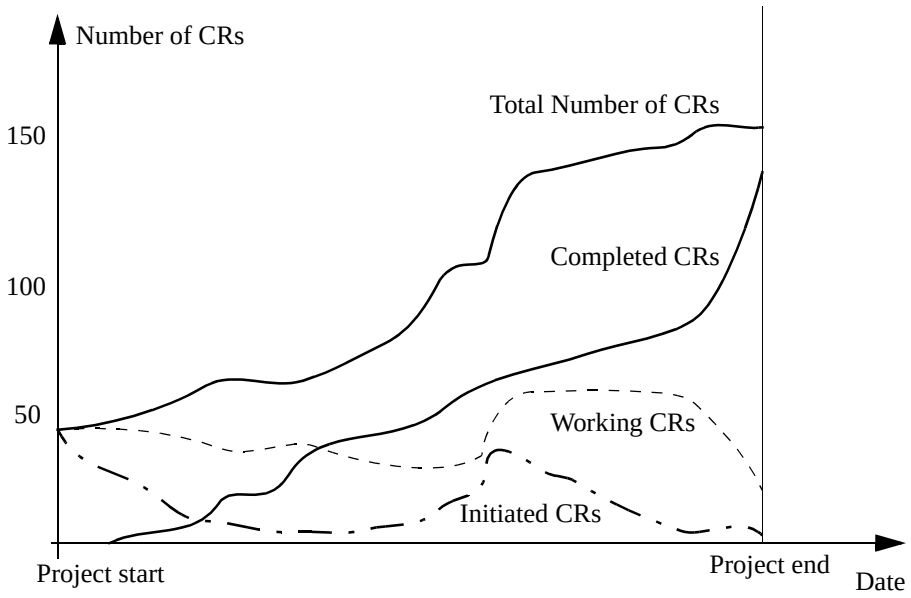

Fig. 6. CR states during a development project life

During the development phase the project managers find very useful information about how many percent of CRs are completed, how many are under the test procedure, how many are open, how many are still in the Initiate state.

4 Conclusion

A long experience of using two different SCM tools with different levels of Change-oriented approaches shows:

- It is a quite challenging job to use a strict change-oriented model, where it is not quite clear which file versions will be included into a software release. Programmers prefer to work directly with files.
- A usage of a change-oriented model is more appropriate for maintenance than pure development.
- It is not possible to achieve an exact mapping between logical and physical changes. There is always a risk that a programmer make changes in a file which cover both registered and not registered logical changes.

- Information from a change-oriented tools can be utilized in the process development - for project follow-ups, planning, quality assurance, etc.
- A SCM tool that (also) manages logical changes is needed for an efficient development and maintenance process for large software systems.
- We need a tool that comprises both approaches - a tool that controls component versions and in the same time support a change-oriented process.

References

1. Ian Sommerville (Ed.), Software Configuration Management - Introduction, *Software Configuration Management ICSE'96 Workshop, Berlin, March 1996, Selected Papers, Springer Verlag, ISB N 3-540-61964-X*, pages 1-7
2. Reinder Conradi and Bernhard Westfechtel, Configuring Versioned Software Products, *Software Configuration Management ICSE'96 Workshop, Berlin, March 1996, Selected Papers, Springer Verlag, ISB N 3-540-61964-X*, pages 88-109
3. Ivica Crnkovic, Large Scale Software System Management, *Ph. D. Thesis 1990, University of Zagreb, Faculty of Electrical Engineering*
4. Walter F. Tichy, RCS - A System for Version Control, S*oftware and Practice Experience*, 15(7):635-654, 1985
5. Ivica Crnkovic, Experience of Using a simple SCM Tool in a Complex Development Environment, *Software Configuration Management ICSE'96 Workshop, Berlin, March 1996, Selected Papers, Springer Verlag, ISB N 3-540-61964-X*, pages 262-263