Middle East University

Proposed Model For Software Configuration Management Theoretical Tool

A thesis submitted in partial fulfillment
of the requirements for the degree of Master of Science
in Computer Information Systems
By

Yousef Salman Al-Shaikhly

Supervisor
Prof. Mohammad Al-Haj Hassan
Middle East University

Co-Supervisor
Prof. SalehAbo Al-Sood
NYIT University

Amman, Jordan
April, 2011

نموذج تفويض

انا يوسف سلمان الشيخلي , افوض جامعة الشرق الاوسط بتزويد نسخ من رسالتي / اطروحتي للمكتبات او المؤسسات او الهيئات او الافراد عند طلبها

التوقيع :

التاريخ : 30/4/2011

Authorization Form

I am Yousef Salman Al-Shaikhly, authorize Middle East University to supply copies of my Thesis / Dissertation to libraries or establishments or individuals upon request.

Signature:

Date: 30/4/2011

Committee Decision

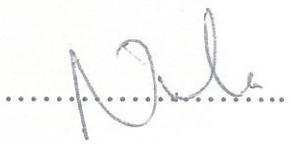
This thesis (Proposed Model For Software Configuration Management Theoretical Tool) was successfully defended and approved on April 12, 2011

Examination Committee Signature

Prof. Mohammad M. Al-Haj Hassan
Professor
(Middle East University)

Dr. AshrafBany Mohammad
Assistant Professor
(Middle East University)

Dr. Nuha Al-Khalili
Associate Professor
Al-Petra University



DEDICATION

This is dedicated to my family, for their love and encouragement.

ACKNOWLEDGMENTS

I would like to express my sincere appreciation to Professor Mohammad Al-Haj Hassan and Professor Saleh Abo Al-Sood for their guidance, support and motivation throughout the work in preparing my Master's Thesis.

I would further like to acknowledge all of the faculty of Information Technology members at the Middle East University for helping and encouraging my efforts especially at the beginning of the thesis. I also thank the reviewers of my thesis Dr. Ashraf Bani Mohammed and Dr. Nuha Al-Khalili for their remarkable comments and remarks, and I would like also to thank the head of CIS department Dr. Hiba Nassiredeen.

Also I would like to thank the Masri's family and Mr. Miqdad Annab (may he rest in peace) for their support at work during the thesis preparation.

Above all, I would like to especially thank my parents for supporting me during the time I was preparing and writing this thesis. Without them nothing of this would have been possible.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

CCB	Change Control Board
CI	Configuration Item
CLI	Command Line interface
CR	Change Request
CVS	Concurrent Version System
ECO	Engineering Change Order
EOID	External Object Identifier
GUI	Graphical User Interface
MCR	Major Change Request
OID	Object Identifier
PR	Problem Report
RCS	Revision Control System
SCM	Software Configuration Management
SVN	Subversion System
TFS	Team Foundation Server
VOB	Versioned Object Base
VSS	Visual Source Safe

ABSTRACT

When an organization develops a software product, the main factors behind the success of that product are the quality and delivery time of the product. During the course of the development project, many changes may arise and affect the initial plans of the development. Software Configuration Management (SCM) is the methodologies for controlling these changes and making sure that changes are implemented in an efficient and timely manner, while preserving the overall quality of the product. Software configuration management is an umbrella activity that controls the changes during the development of the software through all its lifecycle phases, without forcing time-wasting overhead to the process.

SCM controls becomes mostly necessary in a software development environment which has large development teams (especially if the teams are placed in geographically dispersed locations), or in an environment producing software packages for multiple customers, each with his own unique set of requirements.

In software development nowadays, the developers tend to change the system in hand constantly to adapt to external environments changes, changed requirements, extending system's functionality or simply errors correction. If changes were not formally tackled and managed, the environment would be in a case of total chaos. Every developer changes system components as he sees best. Hence comes the importance of software configuration management.

This thesis examines the principles and applications of SCM. SCM concepts, procedures and models based on previous studies which are discussed to familiarize the reader with the SCM process in general. And in the field of automated SCM, Different SCM tools such as CVS, RCS, ClearCase are presented and analyzed for the automation of SCM process, and their limitations and strengths are highlighted. Also in this thesis, we tackle the in-common limitations of SCM tools, and upon those limitations, we construct a proposed model for an SCM theoretical tool that eliminates the limitations observed, and provides better automated approach to SCM.

الملخص

عندما تقوم منظمة بتطويرمنتجات البرمجيات تكون الجودة وفترة ايصال المنتج هما العوامل المؤثرة على انجاح هذا المنتج, وخلال عملية التطوير للمشروع, تطرأ تغييرات تؤثر على الخطة المبدئية لعملية التطوير. أما نظام ادارة تشكيلة البرمجيات هو المنهجية للتحكم بهذه التغييرات والتأكد من أن يتم تطبيق التغييرات بطريقة فعالة وفي الوقت المناسب، مع الحفاظ على الجودة الشاملة للمنتج. وهذا النظام هو المظلة التي تتحكم بالتغيرات التي تطرأ على عملية تطويرالبرمجيات في جميع مراحل دورة حياته، مع ضمان عدم اضاعة الوقت خلال عملية التطوير.

SCM اصبحت ضرورة في عملية تطوير البرمجيات واالتي تتضمن فرق تطوير متنامية العدد (خصوصا اذا كانت هذه الفرق تتوزع جغرافيا في أماكن متباعدة) أو تكون في بيئة انتاج حزم البرمجيات لعدة عملاء, كل منهم مع متطلباته الخاصة والفريدة.

في مجال تطوير البرمجيات في الوقت الحاضر, يميل المطورين الى تغيير النظام المتداول للتكيف مع التغيرات الخارجية,وتغيير المتطلبات ,وتوسعة وظائف النظام أو لمجرد تصحيح الأخطاء.واذا لم يتم معالجة وادارة التغييرات ومتابعتها ضمن بيئة الانتاج, ستعم الفوضى في تلك البيئة. وخصوصا عندما يقوم كل مبرمج بتغيير مكونات النظام كما يراه مناسبا. ومن هنا تكمن أهمية نظام ادارة تشكيلة البرمجيات.

هذه الرسالة تبحث في المبادئ والتطبيقات لنظام ادارة تشكيلة البرمجيات. وايضا تبحث في الإجراءات والنماذج القائمة على الدراسات السابقة التي تمت مناقشتها لتعريف القارئ بماهية نظام ادارة تشكيلة البرمجيات في العموم.

وتدرس الرسالة مبادئ وخطط نظام ادارة تشكيلة البرمجيات. يتك دراسة الخطط والمبادئ والنماذج المتوفرة من قبل الباحثين السابقين في هذا المجال.في مجال عملية اتمتة نظام ادارة تشكيلة البرمجيات,تم دراسة عدة برمجيات توفر اتمتة هذا النظام, وابراز نقاط القوة والضعف فيها مثل RCS, CVS, ClearCase.وايضا تحتوي الرسالة على نموذج مقترح لنظام ادارة تشكيلة البرمجيات نظري لمواجهة نقاط الضعف الملاحظة وتوفر الية افضل لنهج نظام ادارة تشكيلة البرمجيات

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

In the business of producing software that is reliable and of good quality, and competitively worthy among lots of other competitors in the market is considered an exhausting job to do, in which there are many risks to consider. One of those risks and challenges is the controlling of software changes over the duration of the project life.

Changes that occur during the development project could lead to better quality of the produced software, by motivating developers to initiate new ideas that could produce richer software that could compete better in the market. Also those same changes could lead to the termination of the project with failure if not controlled properly.

Some software engineers might consider SCM as a support function and to be an accessory rather than a necessity, but as we will elaborate more about SCM here in this thesis, SCM is a primary function of software development and will help organizations identify potential problems, manage changes, track progress and performance of any product during its lifecycle.

Changes as they occur in the project have a deep effect on the project course after their occurrence. But many factors also intensify the complexity of dealing with those changes and their effect on the project. Like the project size, because the larger the project, the harder it gets to maintain its changing environment. Also the type of the environment could affect the impact of the changes, for example, if the development environment has multiple geographically dispersed developers and if working in parallel on the same project asset is permitted for developers, or if the organization is depending on third-part products to develop subsystems. Also the customers' requirements and interference have a major effect on changes management, since in some packaged products, each customer could have his own requirements of the product alongside his own predetermined deadline for the delivery of the product.

These challenges can cause chaos if there is no right procedure used to control the whole software development process. On any team project, a certain degree of confusion is inevitable. The goal is to minimize the confusion so that more work can get done. The art of coordinating software development to minimize this particular type of confusion is called configuration management. Configuration management is the act of identifying, organizing and controlling the

modifications to the software being built by a programming team. The goal is to maximize productivity by minimizing mistakes. [38]

SCM mainly is a two-phased process, although each organization could have its own customized process in the configuration management filed, it comes down to only two major phases when dealing with SCM no matter how sophisticated and detailed the implementation is. Those two phases are elaborated below:

Two major activities are included in configuration management. One is the management of the development of components and integration of these components to build the complex system that resides on them. Another one is the description of the activity that selects appropriate components to produce a particular instance from related complex systems. Because of the cost and quality benefits of the configuration management for a large and evolving software project, many commercial complete or partial tools have been built for configuration management. [4]

Many organizations continue to apply conventional configuration management methods, then blame this core discipline for failures, or abandon SCM altogether out of frustration. Practitioners of SCM often face constraining resources in their organizations with widespread unawareness of SCM. In the past, budgets were often trimmed of SCM-related costs even before the program or project is funded in a misguided attempt to save money or to even use it as a sort of financial caution for contingencies. In today's environment, this practice is totally unacceptable. SCM has a major role to play in the planning for, procurement and lifecycle of any system or product. [35]

Even in the simplest of projects, every Project Manager knows that a change, no matter how small, in the baseline of a project, is a risk to its schedule, costs, scope and ultimately the final outcome. This is why every project manager must also practice SCM, to control the changes to their product and be able to communicate effectively the consequences of the change (e.g. risks, costs, and schedule). [35]

Hence in this thesis we will discuss the benefits and effectiveness of the SCM process, while elaborating the difficulties and risks that accompany the concept of SCM, alongside the tools that automate the process and standards of SCM, and its strengths and weaknesses.

Also we would unveil a proposed model to tackle the challenges and weaknesses observed in the reviewed SCM tools, as an attempt to gain better SCM automation theoretically. Also we discuss the challenges and risks in implementing SCM in a typical environment and the changes that occur to the nature of the development process along with some best practices to introduce SCM into a software development process. Also in chapter 6, we would discuss the interviews conducted locally and their results on the evaluation of the proposed model.

1.2 STATEMENT OF PROBLEM

In software development companies, different development methodologies are being used, but in general, all configuration items which are also considered as software artifacts are to be managed and controlled throughout all of the development lifecycle.

There are lots of available commercial and open-source tools which support the automation of configuration management cycle across the development project. In this thesis, we first tackle the problem of choosing between those available tools to satisfy the organization's requirements through discussing their main features and workflows, alongside their shortages and limitations.

As a software development company which intends to implement a configuration management plan in their environment, the dilemma would be which software can suite this environment requirements and budget, and which will provide the exact set of features to make the company's development environment more productive and change-oriented.

Also in chapter 4, we propose a model for an SCM theoretical tool, in which we tackle the limitations and shortages observed during the review of SCM tools in chapter 3, and building a proposed model that could eliminate such shortages in those tools to gain better productivity out of SCM.

Also since configuration management is an umbrella activity that would change how software is developed in those companies; we also tackle the problem of changing development environments from typical ones to more sophisticated environment through managing change within the project. This is done by discussing what changes await the companies that are about to implement SCM controls in their environment. This is covered in chapter 5.

1.3 THESIS MOTIVATION

Our motivation is based on the following fact: changes during the development and maintenance phases of a system are the biggest threat to the success of the project in an organization. and not controlling this threat effectively could lead the project to disastrous results. As a part of a software development team, the system engineers face the fear of change in their everyday work, and as we go higher in the managerial levels, the sense of danger of changes increase as they obtain much wider visibility of its effect to the entire project.

We have chosen to tackle the subject of software configuration management because of the following reasons:

1. Because changes are important to the development process
2. Because of the lack of utilization of standards in controlling changes in local market project
3. Because as observed in local companies, the field of Software Configuration Management is still fresh and underestimated mostly. Alongside some challenges in implementing SCM locally through which has limited SCM usage.

1.4 THESIS GOALS

In this thesis we aim to:

- Discuss the available high-level standards and procedures for configuration management and its basic how-to implement a successful configuration management plan.
- Discuss the available commercial and open-source configuration management tools, and its distinguished features sets, also unveiling their limitations to the reader, and emphasizing the fact that limitations still exist in most of the SCM tools when handling SCM.
- Relying on the two previous goals, we intend to develop a model for an SCM theoretical tool that eliminates the shortages found in current tools, and to suggest best practices for implementing configuration management tools in software development environments.

1.5 CONTRIBUTION OF THE THESIS

In this thesis, we discussed different related work relative to the concept of Software Configuration Management (SCM), and constructed a well-structured understanding in the mind of the reader of the world-wide SCM process from multiple viewpoints. Also we analyzed multiple open-source and commercial SCM tools on the market, as to deliver a basic idea of how to automate the concept of SCM, and what to expect from the tool itself.

Alongside the review of each tool, we investigated the limitations in the tools. Depending on those limitations, we proposed a theoretical tool that tackles the common shortages of most SCM tools discussed.

Also we conducted a market study concerning SCM implementation, and provided guidelines and challenges to expect when introducing an SCM tool to a fresh typical software development environment.

1.6 METHODOLOGY

This thesis relied on multiple methodologies, each suitable for a phase in the thesis preparation as illustrated below:

1. I started to investigate the importance of SCM in the software development industry, and I have been convinced that this subject is critical to such industry
2. I searched for literature in this area, and found several important researches in that regards. I read and analyzed the most relevant works in that literature, and made some comparisons and conclusions about the research.
3. I noticed that limitations and shortages were found in tools that offered automated SCM. Therefore, I moved forward to suggest a model that tackles those limitations.
4. In addition to the proposed model, I designed a questionnaire and distributed it across several software development companies in Jordan. I analyzed the answers and drew conclusions related to the fields of baselines, version management and change management (which are the main contribution of our proposed model) to determine whether our proposed model adds value to the industry or not.

1.7 THESIS ORGANIZATION

In addition to this chapter, which introduces the thesis problem and other related ideas, this thesis is organized as follows:

Chapter 2: (the following chapter) provides an overview of the configuration management main concepts as discussed by many researchers, and a set of main components in the software configuration management. Also it discusses an overview of the IEEE standard for configuration management planning and a typical scenario for configuration management lifecycle during the development lifecycle. Also it includes the studies conducted in the local market for software development in Jordan.

Chapter 3: provides an overview of multiple commercial and open-source configuration management tools, some distinguished features will be discussed and shown how a user can benefit from such benefits in a development scenario. Also it will provide the limitations of those tools and build base for the next chapter to develop a more suitable conceptual tool.

Chapter 4: provides the proposed model that will contain counterparts and proposed solutions for configuration management tools limitations in features and workflows, and will provide guidance for parties interested in implementing SCM standards that could aid in the journey of introducing configuration management tools into the company's development environment.

Chapter 5: provides conclusions based on the questionnaires filled by a sample of local market software development companies. Also future works are provided in this chapter.

Chapter 2

Literature Review

Changes are inevitable when a software is built. A primary goal of software engineering is to improve the ease with which changes can be made to software. Configuration management is all about change control. Every software engineer has to be concerned with how changes made to work products are tracked and propagated throughout a project. To ensure that quality is maintained the change process must be audited. A Software Configuration Management (SCM) Plan defines the strategy to be used for change management. [23]

Change management, more commonly called *software configuration management* (SCM or CM), is an umbrella activity that is applied throughout the software process. Because change can occur at any time, SCM activities are developed to (1) identify change, (2) control change, (3) ensure that change is being properly implemented, and (4) report changes to others who may have an interest. [23]

2.1 SCM Introduction

Multiple authors tried to define the software configuration management, According to Bersoff, Henderson and Siegel, the definition of SCM is: SCM, like CM, is defined as the discipline of identifying the configuration of a system at discrete points in time for purposes of systematically controlling changes to this configuration and maintaining the integrity and traceability of this configuration throughout the system life cycle. [6]

According to IEEE standards, the definition of SCM is: 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. [16]

According to Whitgift, the definition of SCM is: Configuration Management (CM) is a collection of techniques which coordinate and control the construction of a system. Many of the principles of CM were developed to enable hardware engineers to design and assemble the components or ever more sophisticated configurations. Some of the principles would have been familiar to the project manager charged with building the pyramids. [37]

According to Wikipedia.org, the definition of SCM is: In software engineering, software configuration management (SCM) is the task of tracking and controlling changes in the software. Configuration management practices include revision control and the establishment of baselines. [55]

According to Bach, Software configuration management is: the development and use of standards and procedures for managing an evolving software system SCM can be viewed as a software quality assurance activity that is applied throughout the software process. [1]

Many software engineers may confuse software maintenance and software configuration management, software maintenance starts after the project has been fully implemented and installed in the customer's site. But software configuration management starts at the beginning of the software development project and continues through all phases of the project.

If a software development company does not establish good SCM plan and procedures to control the activities of its projects, the project will turn into chaos and unexpected changes would take control of the development activities, and might even lead to the project failure.

If changes are not controlled during the start of the project, it takes control of the project and changes become a devastating activity and that's never good. It is very easy for a stream of uncontrolled changes to turn a well-run software project into chaos. For that reason, change management is an essential part of good project management and solid software engineering practice. [23]

The main sources that trigger changes in Software systems development do not only include customers, they also include the users of the system, the developers and market forces, and the reasons those sources suggest changes to the original plan of development include [23]:

- New business or market conditions dictate changes to product requirements or business rules
- New stakeholder needs demand modification of data, functionality, or services delivered by a computer-based system
- Business reorganization causes changes in project priorities or software engineering team structure
- Budgetary or scheduling constraints cause system to be redefined

2.2 Software Configuration Management Main Components

Every system has components that interact together to accomplish the big picture of the whole system. In SCM those main components are the Software Configuration Items (SCI), Baselines, Repository, Version Management and Change Management.

In the sections below, we will discuss in detail each field and how authors explained their roles in completing the SCM cycle.

2.2.1 Software Configuration Items

Configuration Item "also known as Software Configuration Items (SCI) or simply (CI) or Software Artifacts" is a unit or a collection of lower-level items such as software, hardware or both is treated as a single entity for the purpose of configuration management. [10]

CIs are produced by the software development process; they expand to include System specifications, System design, Computer programs (both source code and executable), Test Data and User manuals.

Those CIs are the basic components of any SCM system, the SCM system stores and manages them in its dedicated database. Also the SCM provides mechanisms for accessing, retrieval and update of CIs in a controlled and audited manner.

Some organizations place software tools under the SCM control, because these tools do provide documentations, data and source codes, and as a result, those tools might affect the flow of the development process in case the version of the tool used to produce an under change component is changed. [23]

Those CIs under configuration management control must be uniquely identified and stored in the SCM repository, each object created by the software engineer can be classified into two broad categories: concrete object which only includes the object itself as one unit and composite object which contains multiple objects interrelated to construct the one unit. Both types of objects are treated as single units by the Version Control system (to be discussed later) to avoid unnecessary complexities. [20]

2.2.2 Baselines

A baseline is a checkpoint in the project lifecycle, which can be reached through constantly changing the CIs in the projects repository, until a satisfying state of the product is reached, and hence marked by the baseline.

The IEEE (IEEE Std. No. 610.12-1990) 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. [15]

According to Time Electronic Textbook [31], a baseline is a designation of a snapshot in time of a product or system, with a specification of all identifications of all CIs that are part of it. All CIs included in the baseline have a certain status.

A configuration baseline is characterized by a set of documents, which describes the characteristics, or certain characteristics of a product. This set of documents is formally designated as the configuration reference at a key stage in the product life-cycle, which thus corresponds to a major product definition event. Any change of a product attribute specified or disclosed in this set of documents shall be subject to a formal change procedure involving all the actors and disciplines concerned before it can be taken into account. [16]

Before a set of CIs become a baseline, changes could be made quickly and informally. However, when a baseline is established and approved, changes to included CIs can still be made, but a specific, formal procedure must be applied to evaluate and verify each change.

The following Figure 2.1 illustrates the transformation of software configuration item. The figure shows the most common CIs in software development, a comparison between items' states before and after baselines are defined and the role of the project database in the change management process.

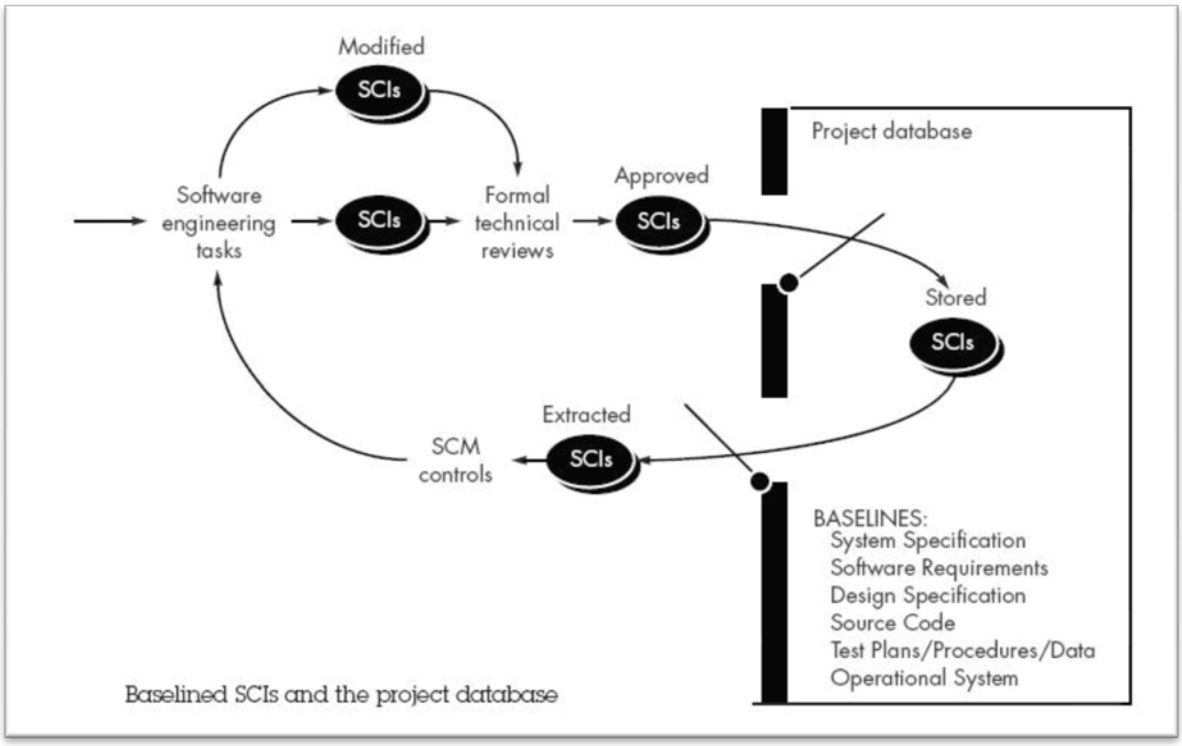


Figure 2.1: Baselined CIs and the project database [23]

According to Figure 2.1, a Software Configuration Item becomes a baseline only after it has been reviewed and approved through formal technical procedures; errors might be found in the in-subject item that need to be corrected before submitting it, and once these corrections are completed, it becomes a baseline. [23]

2.2.3 Configuration Items Repository

According to Sommerville, The configuration repository is used to record all relevant information about system configurations and configuration items. The SCM repository helps assess the impact of system changes and to generate reports for management about the SCM process. As part of the SCM planning process, the SCM repository schema must be defined, along with the forms to collect information to be recorded in the database and procedures for recording and retrieving project information. [27]

Nagel also discussed the concept of the repository in an SCM system; he stated that the repository may simply be a structured directory on a server with each versioned file stored separately, or it may be a database containing entries for the various files in a project. It may even be a complex distributed system that redundantly stores the versioned project all over the world. [19]

The process of SCM relies on the repository residing underneath it heavily. if the architecture of that repository is not sophisticated as the process itself, it will be a burden rather than an asset to the organization implementing it, and it will limit the capabilities of the SCM methodology as a whole. Each object in the project becomes an SCM CI once it is placed under SCM control and is stored in that repository for further review and execution, and hence the repository gains this important role in the SCM process.

Also the repository must be able to store many types of files, as in the multimedia databases abilities of storing multi-type files and presentations; those file types can include documents, diagrams, multimedia files, presentations and many other types that might be produced by the development process.

Each SCM tool has its own structure of a repository, but there is a list of main features and functions that the SCM repository should provide to the users, which includes the following [21]:

1. Data integrity
2. Information sharing
3. Tool integration
4. Data integration
5. Methodology enforcement
6. Document standardization

An SCM repository might have to serve geographically distanced users, as many organizations are increasingly involved in software development efforts that are distributed and decentralized in nature. [21]

In these settings, Software Configuration Management (SCM) becomes a serious challenge and this challenge exhibits itself at several levels. One of the challenges is the issue of distributing large amounts of data in a timely fashion over great distances. [21]

A decentralized SCM repository should provide the following to sufficiently support the process of decentralized software development 1) a central document repository with an audited document content version control 2) powerful document metadata management and versioning 3) sophisticated document authoring management and workflow 4) full-text indexing technology allowing search within document contents 5) powerful role-based security model.[21]

The SCM repository must provide means of recording transactions performed on each CI under its control. The transaction itself must contain the creation date, check-outs and check-ins as minimum. Also it should record the version history of those CIs, it should assist the SCM system

to work as a time machine which can gain access to previous states of a CI and compare it to a later state.

The SCM repository must have a permission distribution mechanism to control which users and groups access what and with what permission, typical permissions assignment for groups and users are: Read, Write, Add Folder, and Manage Permissions.

Users and groups who have read access to a document are allowed to download it for viewing only. A user can choose to open the document or save it to local computer. User can view a version history of all changes to the document, and the user can download previous versions for comparison.

Any changes that a user makes to the document downloaded won't display in the version in the repository unless the user checks out the document, and then checks it back into the repository with the changes.

Users that have only Read permission on a document can view the document and its metadata, but are not allowed to change it. Users that have Read and Write permissions are allowed to change the contents of a document, as well as its metadata.

Another feature that should also be included in the SCM repository is the search feature, a user or a developer could need to search for a file needed, in the content of files or attributes.

The search feature should be consistent with the permissions module for security purposes, so that the search should only display results that the logged in user have access to.

Also a review workflow process could be implemented in the SCM repository, in which the review of a newly-presented CI can be defined to be reviewed by a resource and hence to be approved and published to the repository, and that process should record who approved what for future references. [21]

Those features listed above are the main features of a general SCM repository that could work for a centralized or decentralized software development project, and could support whatever features that the SCM methodology is enforcing.

2.2.4 Version Control

Version control combines procedures and tools to facilitate the management of different versions of CIs created during the software process; through Configuration management multi-directional development is practiced, alternative configurations of the software system are built, through assembling appropriate versions of the objects or components. This is supported by associating attributes with each software version, and then allowing a configuration to be specified by describing the set of desired attributes. [7]

At minimum, the version control systems should be able to provide the following capabilities [23]:

- Project repository integration: stores all relevant configuration objects
- Version management capability: stores all versions of a configuration object (enables any version to be built from past versions)
- Make facility: enables collection of all relevant configuration objects and construct a specific software version
- Issues (bug) tracking capability: enables team to record and track status of outstanding issues for each configuration object

Version management system uses a system modeling approach, a template which includes component hierarchy and component build order, construction rules, verification rules for building software systems from an object repository. A number of different automated approaches to version control have been proposed over the past decades. The primary difference in approaches is the sophistication of the attributes that are used to construct specific versions and variants of a system and the mechanics of the process for construction. [23]

The main idea behind version management is each software object carries an object identifier (OID) that serves to identify it uniquely within a certain context. After the object has been identified in the SCM repository, it becomes easy to attach more information (attributes) to that object. OID then would facilitate the construction of variants of the same program; to construct the appropriate variant of a given version of a program, each object can be assigned an "attribute set " which is a list of features that will define whether the entity should be used when a particular variant of a software version is to be constructed. [10]

Dart discussed the concept of version management by stating that the version management systems is mainly used during maintenance of the history of modifications to the artifacts of a system, including the ability to view a previous version or state of the project, to view differences between two different versions, and to roll back specific changes to recover a previous state of the system. Also the system is supposed to have support for collaborative development, including a conflict resolution mechanism to employ when two or more developers make conflicting changes to the same document, and support for the management of branching of the system under development into two or more parallel lines of development, and selective merging of features of parallel lines of development. [12]

Also Haw focused on the importance of version management in the maintenance phase, by stating that once initial implementation and deployment of a software system is completed, the product must still be maintained. Artifacts such as source code, build files, and documentation need to be preserved to support these efforts.

2.2.5 Change Management

Change management is the main part of every software configuration management system. During this process, any changes which arise during any development or maintenance phase in the software lifecycle are placed under the influence of the protocols that are defined for change control.

Sommerville explained the concept of change management by stating that change management procedures are concerned with analyzing the costs and benefits of proposed changes, approving those changes that are worthwhile and tracking which components of the system have been changed. The change management process should come into effect when the software or associated documentation is baselined by the configuration management team. [27]

Bach addressed the change control in software development by stating that change control is vital. But the forces that make it necessary make it also 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. [1]

An organization intending to manage change that occurs frequently during its projects needs to define change control procedures in a balanced manner. If the organization enforces heavy change control procedures, it will slow performance and developers would abandon the change controls as a whole. But if change control was loose and light, the environment would be a chaos especially in large projects.

2.2.5.1 Change Management Lifecycle

Pressman discussed the change management lifecycle in a very helpful manner, he began the illustration of the lifecycle by stating that when a change necessity arises in the project, it is evaluated then if appears to be doable and adds value to the project, a Change Request (CR) is submitted. The change request is then evaluated to assess technical value, potential side effects, overall impact on other configuration objects and system functions, and the projected cost of the

change. The results of the evaluation are presented as a change report, which is used by a change control board (CCB) which is a person or group who makes a final decision on the status and priority of the change. An engineering change order (ECO) is generated for each approved change. The ECO describes the change to be made, the constraints that must be respected, and the criteria for review and audit. [23]

The object to be changed is "checked out" of the project database, the change is made, and appropriate Software Quality Assurance (SQA) activities are applied. The object is then "checked in" to the database and appropriate version control mechanisms are used to create the next version of the software. [23]

Access and synchronization control flow are illustrated schematically in Figure 2.2. Based on an approved change request and ECO, a software engineer checks out a configuration object for modification. An access control function checks if the software engineer has the authority to check out the object, and synchronization control locks the object in the project database so that no further updates can be made to it until the currently checked out version has been replaced. Note that other copies can be checked-out, but other updates cannot be made. A copy of the baselined object, called the extracted version, is modified by the software engineer. After appropriate SQA and testing, the modified version of the object is checked in and the new baseline object is unlocked. [23]

Prior to CIs becoming a baseline, only informal change control need be applied. The developer of the configuration object (CI) in question may make whatever changes are justified by project and technical requirements (as long as changes do not affect broader system requirements that lie outside the developer's scope of work). [23]

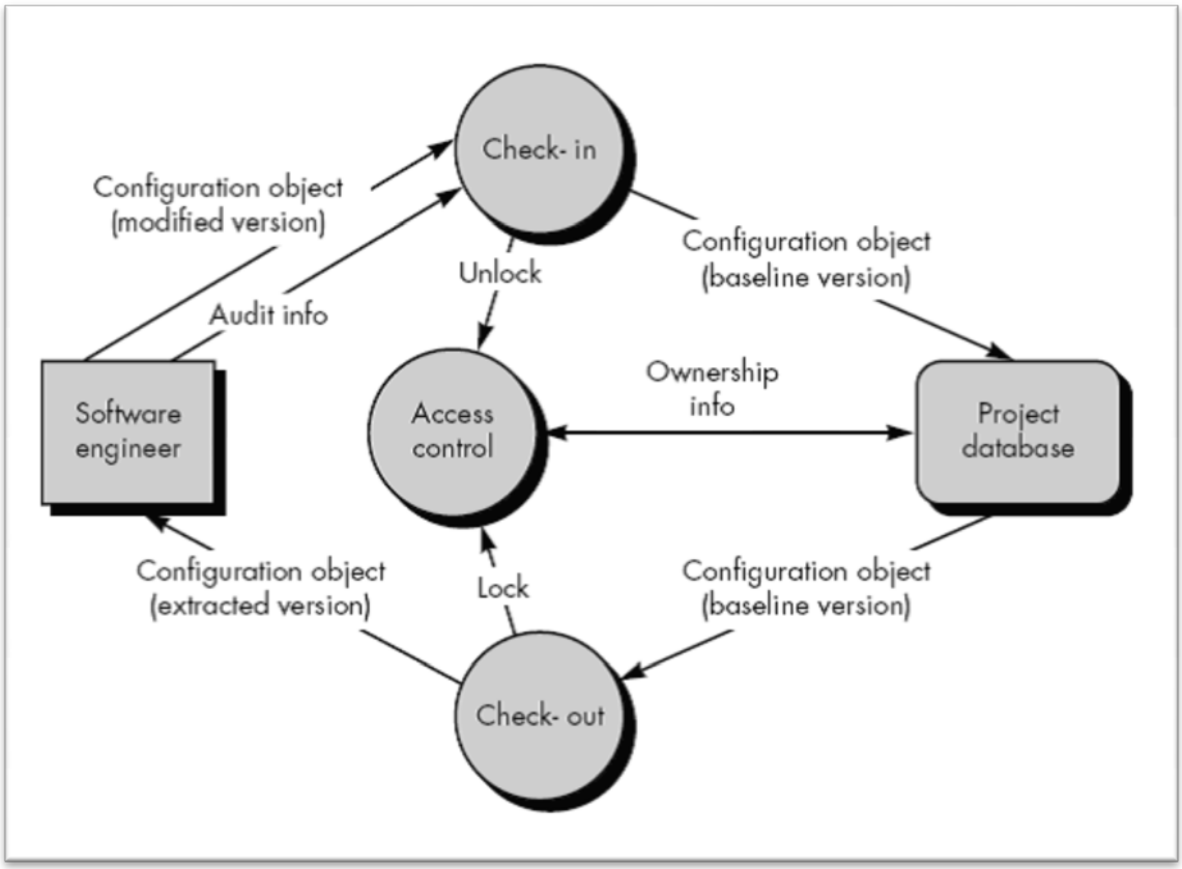


Figure 2.2: Access and synchronization control [23]

Once the object has undergone formal technical review and has been approved, a baseline is created. Once the CI becomes a baseline, project level change control is implemented. Now, to make a change, the developer must gain approval from the project manager (if the change is "local") or from the CCB if the change affects other CIs. In some cases, formal generation of change requests, change reports, and ECOs is distributed. However, assessment of each change is conducted and all changes are tracked and reviewed. [23]

Also Pressman discussed the lifecycle of a change request and the phases it passes through to suggest a change to the project, gain approval and implementing the change. the lifecycle is illustrated in figure 2.3.

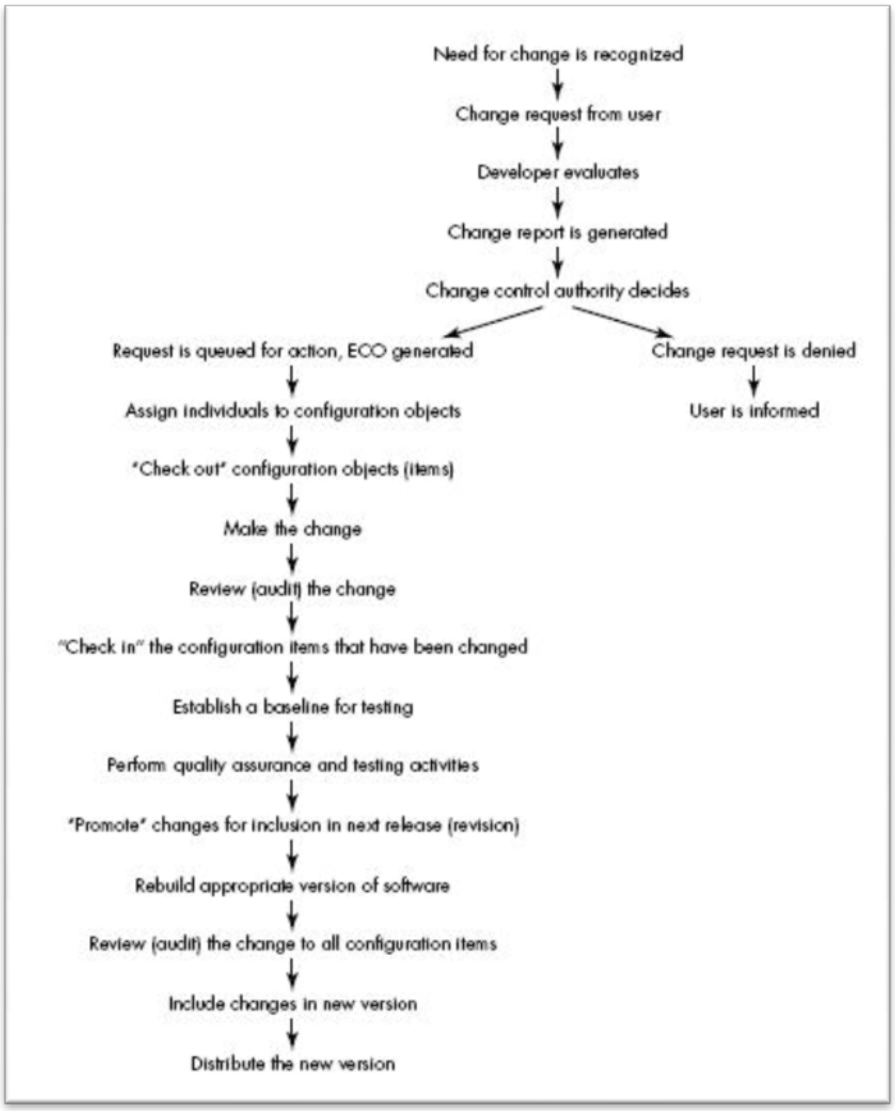


Figure 2.3: The change control process as per pressman [23]

Sommerville also discussed the change lifecycle in his software engineering book. He provided a simple workflow that describes the stages that a change request passes through to complete the change lifecycle. The lifecycle as pointed by Sommerville is illustrated in figure 2.4.

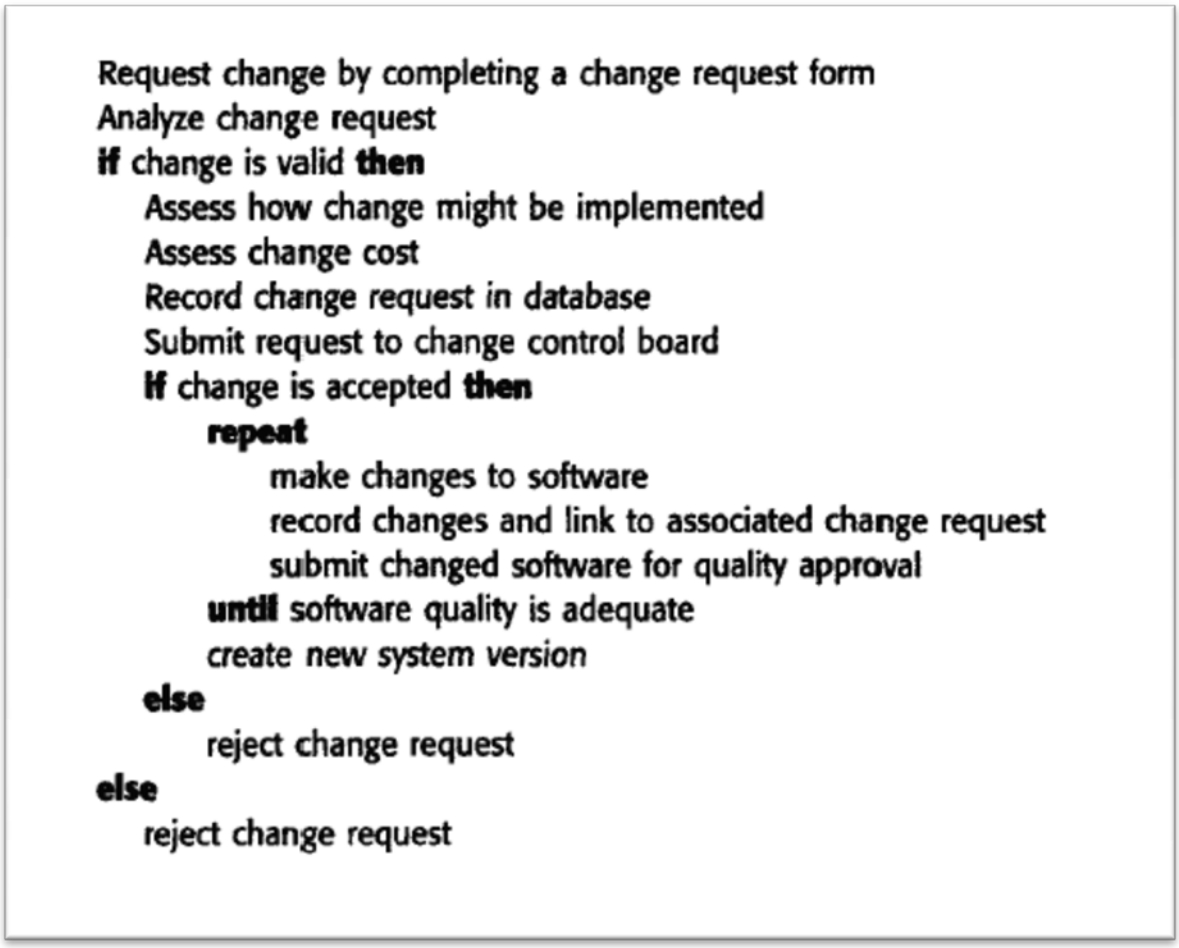


Figure 2.4: the change lifecycle as per sommerville [27]

As Figure 2.4 illustrates, the process of change control begins with the Change Request form being filled to provide information regarding the change at hand. The form should provide information about the change itself, the affected CIs, and the CCB decision and recommendations.

Once a change request form has been submitted, it should be registered in the configuration database. The change request is then analyzed to check that the change requested is necessary. For valid changes, the next stage of the process is change assessment and costing. The impact of the change on the rest of the system must be checked. This involves identifying all of the components affected by the change using information from the configuration database and the source code of the software. [27]

A change control board (CCB) should review and approve all change requests unless the changes simply involve correcting minor errors. The CCB considers the impact of the change from a strategic and organizational rather than a technical point of view. The board should decide

whether the change is economically justified and should prioritize the changes that have been accepted. [27]

2.2.5.2 Software Change Request (CR)

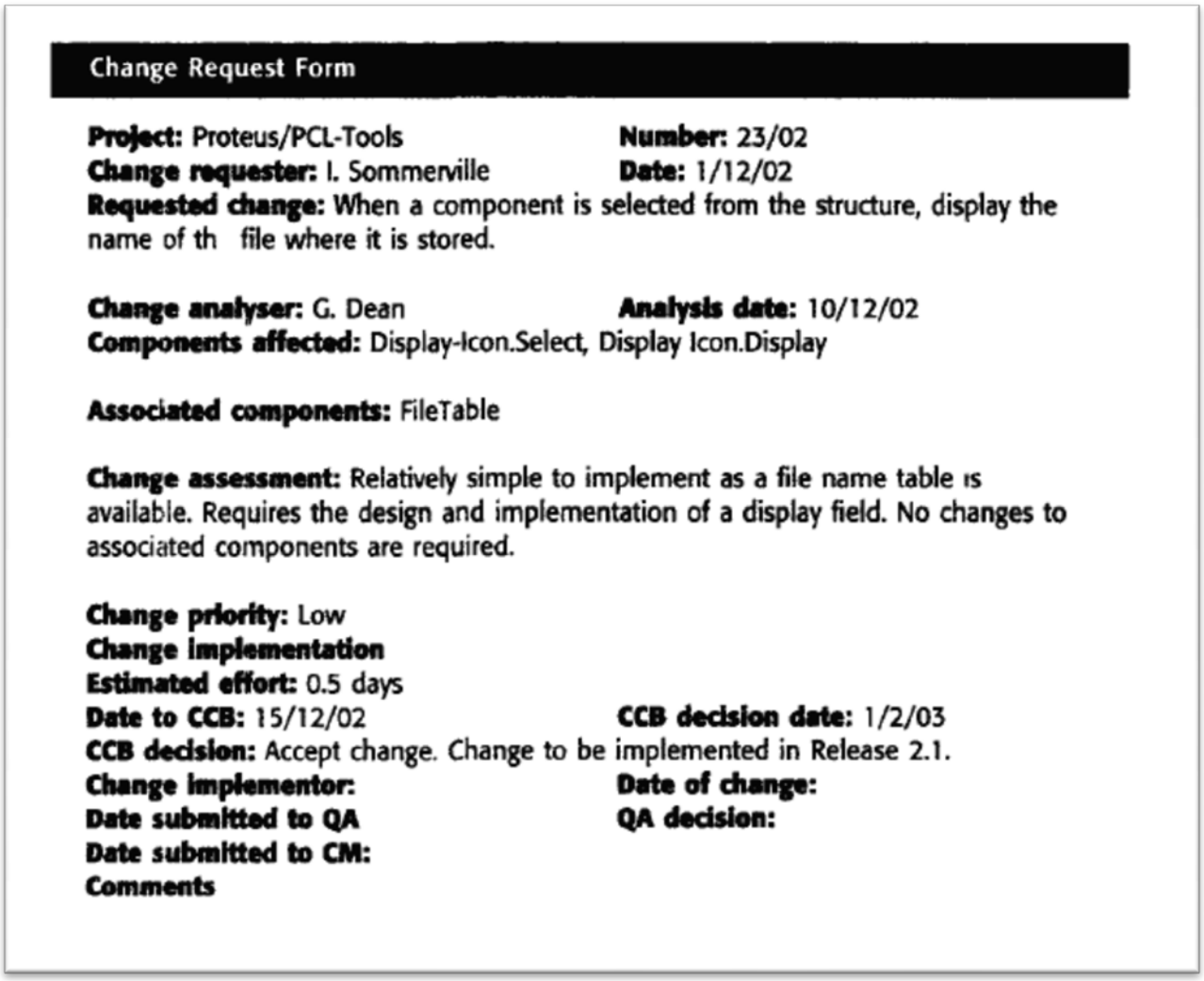
Change requests are the only way to ask for a change in the course of the development, most of the sophisticated and management-focused development companies enforce the usage of those requests to initiate a change process no matter how small the change is. A change request can be defined as a document that describes the requested change and why it is important; it can originate from problem reports, system enhancements, customers, changes in underlying systems and senior management.

The change requests are evaluated by an authorized person or group as discussed before. Therefore, it is important that change requests are well established and filled before being submitted. The CCB is the filter point at which unnecessary change requests are rejected.

Sommerville provided a sample change request to illustrate the fields essential to be provided before submitting a change request. The form is illustrated in Table 2.1.

As illustrated in the form, the change request must identify the project subject to change, the change requester, date of request and a description to the change intended. Also it has fields for change evaluation, which includes the evaluation performer, date of evaluation and affected artifacts by the change.

And finally it includes a section for CCB decision of the change itself, which includes fields to clarify the change priority, the implementation methodology, CCB decision and date of the decision.



Change Request Form

Project: Proteus/PCL-Tools **Number:** 23/02
Change requester: I. Sommerville **Date:** 1/12/02
Requested change: When a component is selected from the structure, display the name of th file where it is stored.

Change analyser: G. Dean **Analysis date:** 10/12/02
Components affected: Display-Icon.Select, Display Icon.Display

Associated components: FileTable

Change assessment: Relatively simple to implement as a file name table is available. Requires the design and implementation of a display field. No changes to associated components are required.

Change priority: Low
Change implementation
Estimated effort: 0.5 days
Date to CCB: 15/12/02 **CCB decision date:** 1/2/03
CCB decision: Accept change. Change to be implemented in Release 2.1.
Change implementor: **Date of change:**
Date submitted to QA **QA decision:**
Date submitted to CM:
Comments

Table 2.1: Change Request most common fields [27]

In figure 2.5, the phases that a CR in any SCM system passes through is shown, a Change Request passes through different states during the development process. When a CR is created it is in the state Initialized. During the work sessions it passes through other states, such as Experimented, Implemented, Tested and reaches the Terminated state. The CRs integrated in a product release are in the state Released. [31]

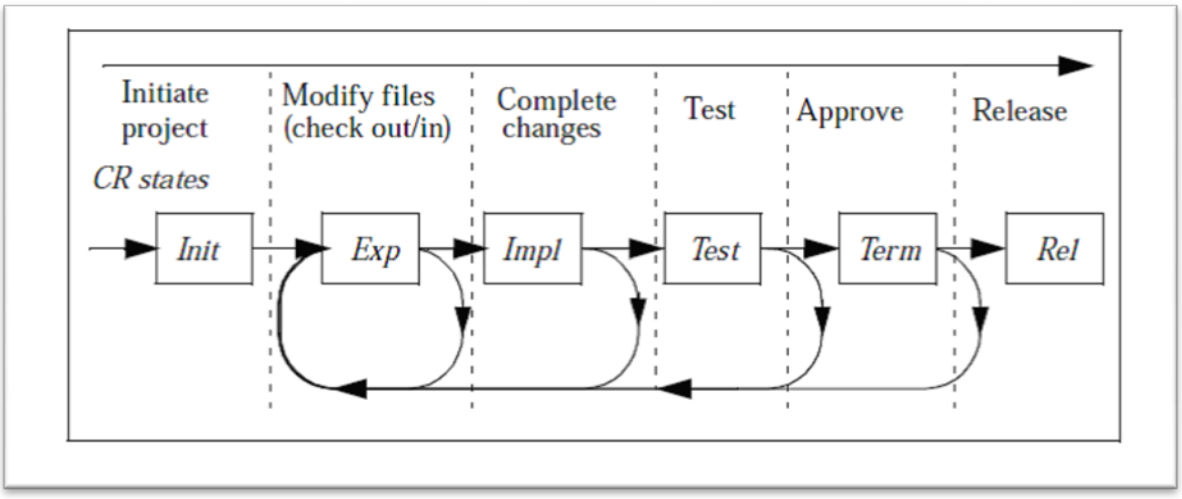


Fig 2.5: Change Requests in a development process [31]

The request is for changes to be made to a section of code or many related parts of the product such as a configuration. The scope of the change is user-defined; the SCM keeps a record of the request and what is affected by the change. The request is assigned a unique name. Once the form is completed with details of all consequences (such as which components will be affected), the form can be electronically mailed to the SCM manager who can mail back an approval or rejection to the software engineer. If approved, engineers are assigned to make the change, and those engineers are given access to the repository based on the change request assignment. [11]

Normally when a CR is received, the first activity is to record it in a CR Register. The CR register could be an Excel Sheet or software like PMPal that facilitates CR Register functionality. The CR Register is the main tool for tracking all CRs to resolution- that may be rejection of the CR or implementation. All CRs received would be entered in the CR Register and would be tracked through to closure. Then the Project Manager analyzes the CR. The main goal of such registration is to keep track of relationships between change requests and configuration items. [18]

Change requests represents a set of transactions of changes to CIs under SCM control, a status report for changes in progress, and an audit trail of changes done and changes rejected. The change request is marked as completed once the changes are implemented. Once the changes are tested and approved, the affected components can then be passed to the approved configuration and this step represents the end of the CR lifecycle.

2.3 Software Configuration Management Plan Procedures

SCM is an umbrella set of activities that need to be planned thoroughly before injecting it into a development process. In this section we provide information about the SCM planning main procedures that will form the whole SCM plan for the organization. The activities included in those procedures require certain information to be provided, and this information identifies all functions and tasks required to manage the configuration of the software system. [29]

The IEEE standards organization produced the most widely-accepted standard for SCM, and most of the tools refer to its activities and lifecycle as the base of the tool implementation. Even though the standard is relatively old, it is still active and newly-introduced SCM tools still use it as their main reference. Here in this section, we discuss the plan introduced by IEEE and its main procedures. [16] [15]

The standard is in the form of a plan. A plan that is meant to be developed by any interested organization for implementing SCM system in their environment, either in an automated methodologies (using an SCM tool), or by handling it using manual procedures (mostly paper work).

The main contribution of the standard is the activities that are undertaken to implement the SCM plan in an organization. Those activities do not describe a tool, only procedures which can be used to build a tool for automation. In this section we address mainly those procedures.

SCM activities are traditionally grouped into four functions: configuration identification, configuration control, status accounting, and configuration audits and reviews. The information requirements for each activity and the activities main purposes in the SCM plan are identified in the sections below.

2.3.1 Configuration Identification

The first step in the SCM planning is Configuration Identification, which is the means to Identify and document the functional characteristics of a Configuration Item and to ensure that the product is visually distinguished from like items. [16]

Dart discussed this part by stating that an identification scheme is needed to reflect the structure of the product. This involves identifying the structure and kinds of components, making them unique and accessible in some form by giving each component a name, version identification, and configuration identification. [11]

SCM plan should record the items to be controlled, the project CIs, and their definitions as they evolve or are selected. It also should describe how the list of items and the structures are to be maintained for the project. [16]

The CIs identification process has small activities to be tackled in order to accomplish the bigger picture; first the organization should provide a scheme for naming CIs. SCM should specify an identification system for assigning unique identifiers to each item to be controlled. It shall also specify how different versions of each CI are to be uniquely identified. Identification methods could include naming conventions and version numbers and letters. [16]

SCM should describe the methods for naming controlled items for purposes of storage, retrieval, tracking, reproduction, and distribution. Activities may include version marking, labeling of documentation and executable software. [16]

Subcontracted software, vendor proprietary software, and support software may require special identification schemes and labeling. [16]

SCM should also identify the controlled software libraries for the project and describe how the code, documentation, and data of the identified baselines are to be physically placed under control in the appropriate library. For each library the format, location, documentation requirements, receiving and inspection requirements, and access control procedures shall be specified. [16]

Configuration Identification is the foundation for all Configuration Management activities, and the starting activity of each SCM cycle, Typical SCM tasks relative to Configuration Identification are [16]:

- Define product structure and select sub-elements to be managed
- Assign unique identifiers
- Select configuration document types & formats
- Define product attributes, interfaces, details in configuration documentation
- Conduct review and coordination of configuration documentation and, if required, obtain customer review and approval
- Establish a release process; release configuration documentation and authorize its use
- Baseline configuration documentation for internal design control and, as applicable, for customer configuration change management
- Assign serial and lot numbers as necessary to differentiate individual units (serialization) and groups of units or bulk material
- Accomplish marking or labeling of products and documentation with applicable identifiers, enabling correlation between the product, configuration documentation and associated data

2.3.2 Configuration Control

Configuration Control is the means to assess and record changes to the functional characteristics of a Configuration Item, revisions to the Configuration Item definition data and monitoring such changes during preparation, review, approval and implementation. Configuration control activities request, evaluate, approve or disapprove, and implement changes to baselined CIs. Changes can include both error correction and enhancement. [16]

Dart discussed this part by stating that the main purpose of the configuration control procedure is to control the release of a product and changes to it throughout the lifecycle by having controls in place that ensure consistent software via the creation of a baseline product. [11]

The SCM Plan shall specify the procedures for requesting a change to a baselined CI and the information to be documented for the request. As a minimum, the information recorded for a proposed change shall contain the name(s) and version(s) of the CI(s) where the problem appears, originator's name, date of request, indication of urgency, the need for the change and description of the requested change. [16]

Additional information, such as priority or classification, may be included to clarify the significance of the request and to assist in its analysis and evaluation. Other information, such as change request number, status, and nature, may be recorded for change tracking. [16]

Also the plan shall specify the analysis required to determine the impact of the proposed change and the procedures for reviewing the results of the analysis. Changes should be evaluated according to their effect on the deliverable and their impact on project resources. [16]

The SCM plan shall identify the configuration control board (CCB) and its level of authority for approving proposed changes. A CCB may be an individual or a group. Multiple levels of CCBs may be specified, depending upon the degree of system or project complexity and upon the project baseline involved. [16]

Finally the SCM Plan shall specify the activities for verifying and implementing an approved change. The information recorded for the completion of a change shall contain the associated change request, the names and versions of the affected items, verification date, release or installation date and responsible party, the identifier of the new version. [16]

Typical SCM tasks relative to Configuration Control are [16]:

- Identify need for change or variance
- Document each request of change or variance and assign identifiers
- Evaluate each change and variance
- Coordinate with affected areas of responsibility
- Classify each request and establish affectivity
- Disposition of each request, obtaining required approvals

- Plan change implementation
- Implement change and verify reestablished consistency of product, documentation, operation and maintenance information, services and training

2.3.3 Configuration Status Accounting

Configuration Status Accounting is the means to record and report product configuration evolution history, Change processing and implementation status and Product documentation status and history. [16]

Dart discussed this part by stating that the status accounting primary output is the recording and reporting the status of components and change requests, and gathering vital statistics about components in the product. [11]

The following minimum data elements shall be tracked and reported for each CI; its initial approved version, the status of requested changes, and the implementation status of approved changes. The level of detail and specific data required may vary according to the information needs of the project and the customer. [16]

In the SCM plan, the types of the reports are to be specified, alongside the types of information to be collected, and the users that should have access to those reports also should be defined.

Typical SCM tasks relative to Configuration Status Accounting are [16]:

- Identify and customize information requirements
- Implement a product technical status related information system
- Capture and report information about:
    - Product configuration status
    - Configuration documentation
    - Current baselines
    - Historic baselines
    - Change requests / proposals
    - Implementation of changes
    - Change proposals
    - Change notices
    - Variances
    - Replacements by maintenance action
    - Configuration verification and audit status
- Provide availability and retrievability of data consistent with needs of the various users

2.3.4 Configuration Verification Review and SCM Audit

Configuration Verification Review and SCM Audit is the means to assess that the final product design is completely documented, the final product design satisfies all performance requirements of the specification(s), the initially produced product meets the design disclosure of its governing documentation, and the Configuration Management assets are sufficient to achieve that goals. [16]

Dart discussed this part by stating that this stage is achieved through validating the completeness of a product and maintaining consistency among the components by ensuring that components are in an appropriate state throughout the entire project life cycle and that the product is a well-defined collection of components. [11]

Typical SCM tasks relative to Configuration Verification Review and SCM Audit are [16]:

- Verify the product within normal course of process flow
- Assure consistency of released information and production / modification information
- Conduct formal SCM audits in-house or at subcontractors / suppliers when required
- Review performance requirements, test plans, results and other evidence to determine that the product performs as specified, warranted and advertised
- Perform physical inspection of product and design information; assure accuracy, consistency and conformance with acceptable practices
- Conduct formal SCM audits in-house or at subcontractors / suppliers as scheduled
- Record discrepancies; review to close out or determine action; record action items
- Track action items to closure via status accounting

2.4 Software Configuration Management in Local Market

In today's software development environments, software configuration management (SCM) has proved its worthiness. But unfortunately in Jordan's software development companies, the concept of configuration management is new to companies, and is considered as an accessory rather than a necessity.

In this section, we present the methodologies applied in local companies to control changes as they occur in projects, types of SCM tools applied in local companies and the challenges of the software configuration management process adaptation.

2.4.1 Local Interviews

We conducted several interviews with companies ranging from small to large sizes, and with employees ranging from developers to senior managers to gain knowledge of how SCM concepts are implemented in local companies.

The interviews collected facts about the local software development companies regarding their processes and procedures in managing changes during their projects development. Also the interviews aimed to gain better understanding of the SCM and VCS tools implemented currently in the local market, and how they are utilized.

We have concluded from the interviews that local companies face multiple challenges while attempting to implement SCM procedures, hence most local companies tend to have manual procedures to deal with changes (as will be illustrated later in this section).

The interviews were conducted in the form of questionnaires, and mainly contained three parts. The first part was to get familiar with the organization's nature in terms of size, projects size, development team and project hierarchy.

The second part was to get familiar with the processes implemented by the organization to deal with change during the lifecycle of their projects, and how they manage, evaluate and approve changes in hand. Also it aimed to gain information regarding the version management and configuration management software tools implemented in the organization (if any existed).

The third part aimed to evaluate the theoretical proposed model main features against the typical SCM tools behavior to gain feedback of our proposed model. Those interviews are included in appendix A in this thesis.

2.4.2 Manual SCM Procedures in Local Market

As observed in local software development companies through the formal interviews, the managerial concepts regarding the process of development is handled manually and in particular the software configuration management process. The change process is handled by paper work, a formal change request template is passed to the developers concerned in the subject, and a project document controller handles the coordination process of the changes.

So whenever a developer needs to change configuration item, he needs to fill a form and pass it to his superior, who will pass it to the decision maker for evaluation. The structure for change processing in manual systems is not clear, so basically in most cases there will be no change control board, and the decision maker will be the project manager (or even might be the team leader).

If the project manager needs any comments or advice regarding the change request between his hands, he will have to contact the appropriate person to seek help. Sometimes this person is from quality assurance, upper management or a software engineer.

The structure or workflow for change management implemented in the local market is sufficient for the size of projects that those companies have to deal with, which are small to medium size in most cases, and to the number of developers assigned to that project.

Although the SCM process proposed guidelines by IEEE does provide well-defined set of rules to handle the changes and builds of a software in a timely fashion, but we found that those set of rules are not as easy to implement in all cases as will be shown through the rest of this section.

2.4.3 Types of Software Configuration Management Tools Implemented Locally

Most of software development companies in Jordan use open-source version management tools with its minimal functionalities; most of them use subversion or CVS and rarely use RCS.

Open-source softwares are free, and hence are largely used in low budget projects to minimize the cost, which is the main factor in promoting softwares in local market. But as known worldwide, open source softwares tend to have limited functionalities and does not provide as much reliability as commercial softwares.

In the local market the main concept of using the version management tools is gain control over revisions of source files, other project related files are placed elsewhere and not under version management control, and is tracked manually for changes.

Other features of version management tools like branching and release management is not utilized by the local market implementers. They rather depend on outsourcing software to manage release operations, build operations and tend to branch at minimum or even never branch.

When it comes to more advanced Software configuration management tools, those were found rarely in local market software development companies, and wherever they were found installed, many factors interfered with the control and scope of the installed tools.

Some of the companies relied on the development language to select an SCM tool. For example, if the company uses a .NET framework to develop softwares, they would rather use Microsoft's Visual Source Safe or Team Foundation Server as their SCM tool in action. But the usage of those tools was kept at minimal since only the database model, version management model and team collaboration features were utilized, but other features were simply deactivated.

The reason behind such endeavor is that the configuration management modules come as a take-it or leave-it package, you either implement the whole cycle, or abandon it all. And since those companies would rather rely on their predefined process for implementing configurations, they have abandoned the lifecycle proposed in the tools and relied on their own manual procedures.

2.4.4 Cost of Ownership

The cost for obtaining the service of a commercial software configuration management tool varies a lot between the variety of tools, starting from 500$ per user for Microsoft's Team Foundation Server, and goes as high as 4,380$ for a floating license of IBM's ClearCase.

Also there are some extra expenses to consider when soliciting a software configuration management tool, which are the hardware requirements, and the extra labor that is needed.

The hardware requirements differ from one tool to another. Some tools require a powerful server to deploy on, and others require only a simple workstation for deployment. Also the network traffic may be an issue to consider, since some tools depend heavily on online transactions between the developers' workstations and the deployment server, which could utilize the network switches in the organization and require an upgrade to the network hardware.

Also most of the SCM tools available require a part-time administrator to perform the tasks of backup and other maintenance activities, and with reasonable training for that administrator.

While some heavy-weight SCM tools, like ClearCase, require a full-time administrator for each certain number of users, and that administrator would be performing backup and other maintenance activities, and diagnostic responsibilities with lots of training to the administrator.

2.4.5 Challenges of Implementing SCM in Local Market

Software development industry in Jordan is an exponentially evolving industry, and has been improving and introducing new technologies and methodologies. But when we conducted the interviews with multiple companies, we concluded that the files in hands "Software Configuration Management" has not been utilized in the development environments yet.

The reasons behind such delay in presenting this methodology to control software development projects are presented in this section.

The first and main reason is financial limitations, because those projects are intended to profit the companies undergoing it, and because the size of projects developed in local market is relatively small comparing to other countries in Europe or USA, and because tools for SCM is licensed per user involved in the project (as described in a later section). All of these are factors that kept the use of SCM in local market currently financially unfeasible.

Most of local market development companies are considered to be small sized, which is that it contains less than 50 employees according to a study conducted in Jordan's local market [13]. In such companies the idea of introducing managerial software that will cost thousands to ease the flow of changes during the development lifecycle of a project is financially risky.

Even in larger companies here in Jordan, the introduction of such software is considered risky, because of the nature of the projects implemented in local market, and because of the size of the teams that work on larger projects, which could not go further than 100 developers per project. And this number of developers can still be easily managed through manual procedures (as produced in later section) according to project managers of some large companies that we have interviewed. Also SCM tends to raise the bar of responsibilities for each user role involved in the project, and hence raising the cost of labor, because the companies implementing a full SCM lifecycle will have to hire more people to cope with the extra tasks and newly introduced roles in the project.

Far from the financial risks, the companies which introduced a certain SCM software into their framework, have encountered an increased overhead and extra documentation to handle, those are the results of the umbrella activities of SCM that controls the project flow during all phases of development. Local market companies consider these extra activities to be useless overhead, and to be rather discouraging to the employees working in the project.

The overhead activities introduced by the SCM lifecycle is meant to control the change that occurs during the development phases, and to control multiple configurations of the same software in hand, but with such level of control to be imposed, comes a level of bureaucracy to go through, and this bureaucracy is not only forced on internal users, but also on all involved parties outside the company developing the software, including the customer himself.

In local market, this level of bureaucracy is found hard to enforce, especially on the customer and involved users of the system, and since no companies in local market depend on geographically dispersed teams to develop a system, and since communications are easy within the site of the company between the members of the project team, such level of bureaucracy is considered a burden rather than an actual beneficial routine.

To gain full control over the project's changes, a company introduces the SCM lifecycle represented by a tool, but the tool cannot operate by itself, it needs an administrator, and lots of information (documentation) to process. And those two newly added perspectives are also considered a burden on the company, extra roles in the project, and extra documents to manage.

Chapter 3

SCM Tools Evolution

In the early 1980s need for versioning objects arose, and thus initiated the idea of version control systems (VCS), in the beginning the main purpose was to label objects with identifiers for tracking purposes and for retrieval in later stages, and then the requirements got heavier and expanded the scope of such systems. Later in the 1990s, a transformation happened to the version control systems to include more managerial concepts and transformed the systems to software configuration systems (SCM). [20]

The first generation of SCM tools were known as Version Control Systems, the main concept behind VCS was to formalize processes based on tracking revisions of system designs. This system of control implicitly allowed returning to any earlier state of the design, for cases in which engineering dead-end was reached in the development of the design. Revision control is any practice that tracks and provides control over changes to system related artifacts. [54]

Then after the widespread of VCSs, additional related features came to the surface, such as those related to change management, and thus produced the SCM systems. The traditional software configuration management (SCM) process is looked upon by practitioners as the best solution to handling changes in software projects. It identifies the functional and physical attributes of software at various points in time, and performs systematic control of changes to the identified attributes for the purpose of maintaining software integrity and traceability throughout the software development life cycle. [25]

We begin our illustration of SCM tools with the introduction of VCS tools which are the first attempts to automate software configuration management. We discuss three VCS tools, which are RCS, CVS and Subversion. Then we discuss five software configuration management tools that automate the process and procedures of SCM, which are Perforce, ClearCase, Team Foundation Server, Visual Source Safe and Changeman.

3.1 Software Configuration Management Tools

In this chapter we discuss three tools for VCS, which are (ordered by creation date) RCS, CVS and Subversion. Also we discuss five tools for SCM which are Perforce, ClearCase, Team Foundation Server, Visual Source Safe and Changeman.

The criteria we used to choose those tools from the widely available tools is first popularity, since those tools are the most tools deployed in software development environments world-wide. Also those tools are the tools that were reviewed properly and discussed thoroughly

Also the Table 3.1 below shows the list of available SCM tools and differentiates between open-source and commercial tools.

Tool Name	Commercial	Open Source
+1CM	●	
AccuRev SCM	●	
Aegis		●
AllChange	●	
BCS		●
BriefCase Toolkit		●
CCC/Harvest, CCC/Manager, CCC QuikTrak	●	
ClearCase	●	
CM Synergy	●	
CMF	●	
CMS and MMS	●	
CMVC	●	
CMZ	●	
Code Co-op	●	
CONTROL-CS	●	
Corporate RCS	●	
CVS		●
Disciplined Software Management	●	
DRTS	●	
DSEE	●	
Emacs		●
Endevor Workstation	●	
ExcoConf	●	
FtpVC	●	
GNU CSSC		●
HOPE	●	
ICE		●
MainSoft Visual SourceSafe for UNIX	●	
Metrowerks Visual SourceSafe for Macintosh	●	
NeumaCM+	●	
ODE		●
PCM	●	
PERFORCE	●	
PRCS		●
PVCS	●	
QEF Software Process Automation System	●	
Quma Version Control System (QVCS)	●	
RAZOR	●	
RCE	●	
RCS		●
SABLIME	●	
SCCS		●
Serena ChangeMan DS	●	

Tool Name	Commercial	Open Source
Serena ChangeMan ZMF	●	
ShapeTools		●
Software Configuration Library Manager (SCLM)	●	
Software Manager	●	
Source Code Manager	●	
SourceOffSite	●	
StarTeam	●	
Subversion		●
TeamConnection	●	
TeamSite	●	
TeamWare	●	
TLIB	●	
TRUEchange	●	
Visual SourceSafe	●	
VisualEnabler	●	
Voodoo	●	

Table 3.1: List of Available SCM Tools

3.1.1 Common Vocabulary

Before we investigate into the evolution of various version control systems and their mechanisms, it is important to understand some of the terminology that is used in versioning. Below is a list of terms commonly used by SCMs.

Baseline

An approved revision of a document or source file from which subsequent changes can be made. [25]

Branch

A copy of the project, under version control but isolated, so that changes made to the branch do not affect the rest of the project, and vice versa, except when changes are deliberately "merged" from one side to the other. [45]

Checkout

A check-out (or co) creates a local working copy from the repository. A user may specify a specific revision or obtain the latest. [25]

Commit

To make a change to the project; more formally, to store a change in the version control database in such a way that it can be incorporated into future releases of the project. [45]

Conflict

What happens when two people try to make different changes to the same place in the code. [45]

Merge

To move a change from one branch to another. This includes merging from the main trunk to some other branch, or vice versa. [45]

Repository

A database in which changes are stored. Some version control systems are centralized: there is a single, master repository, which stores all changes to the project. Others are decentralized: each developer has his own repository, and changes can be swapped back and forth between repositories arbitrarily. [45]

Revision

Also version: A version is any change in form. In SVK, a Revision is the state at a point in time of the entire tree in the repository. [25]

Update

To ask that others' changes (commits) be incorporated into your local copy of the project; that is, to bring your copy "up-to-date". This is a very common operation; most developers update their code several times a day, so that they know they are running roughly the same thing the other developers are running. [45]

Working copy

A developer's private directory tree contains the project's source code files, and possibly its web pages or other documents. A working copy also contains a little bit of metadata managed by the version control system, telling the working copy what repository it comes from, what "revisions" of the files are present, etc. [45]

3.1.2 Common VCS Functionalities

All Version Control System (VCS) tools have some features in common, a good VCS has the following features [26]:

- Backup and Restore. The team can save files as they are edited and have the facility to jump to a previous version.
- Synchronization. Allows the development team to share your source code files and update your codebase with the latest version.
- Undo Changes. Changes anyone makes to the code can be undone by going back to a version that was committed in the past.

- Track Changes. As files are updated, the developer can leave messages explaining why the change happened; this enables the team to see how and why the code evolved over time.
- Track Ownership. The developer can tag changes with the name of the person to track who made the changes.
- Branching and merging. This is where the project manager can branch a copy of the code into a separate area and modify it in isolation (tracking changes separately). Later, the manager can merge the work back into the original codebase.

3.2 Revision Control System

In the early stages of Version Control Systems (VCS) systems development and modeling, the main focus was to establish guidelines and tools to implement versioning mechanisms and revision control upon the software artifacts (also known as CIs), and the task of keeping a software system consisting of many versions and configurations well organized.

In the 1980s, a VCS system was introduced in the name of Revision Control System (RCS), which is a set of UNIX commands tool that was built to enable the user to manage changes to the project's assets in an efficient manner. It emphasized on revisions of source code, documentation and test data and stored them in its own repository and made them accessible to users while maintain a record of changes that occurred to those assets. [54]

RCS was first released in 1982 by Walter F. Tichy while he was at Purdue University as a free and more evolved alternative to the then-popular Source Code Control System (SCCS). It is now part of the General Public License (GNU) Project but its maintenance (given to Purdue University) has stopped with the last version 5.7 released in 1995. [54]

The tool was a bit primitive, but it provided the development team with the automation of storing, retrieval, logging and identification of revisions, alongside the selection mechanisms for composing configurations.

Revision Control System (RCS) provide a database-like storage system for file versions. These systems focus on storing revisions of a file in an efficient manner, and minimizing storage space. Specific information to each version is associated with each version including version date, version number, the person who stored the version, and a log entry entered by the user describing the version changes. [40]

3.2.1 RCS in Operation

When a file (let's name it Intro) is created for the first time in the code editor, the developer working on it needs to check-in the file into the repository under the RCS control; the RCS

engine creates a new revision group with the contents of the file as the initial revision (numbered 1.1) and stores the group into the file “Intro,v”, the v stands for the version of the file which will be included in the file name, files which does not end with a version is a working file, and by checking the working file of a revision group, the developer will invoke the latest version of the file. Unless told otherwise, the command deletes “Intro”. It also asks for a description of the group. The description should state the common purpose of all revisions in the group, and becomes part of the group’s documentation. All later check-in commands will ask for a log entry, which should summarize the changes made. (The first revision is assigned a default log message, which just records the fact that it is the initial revision.) [30]

A feature in RCS named “strict locking feature” can be set by the administrator, this feature is mostly used in a production environment and will ensure that the developer locks the file during the check-out operation and then changing the file and check-in it again to the repository, this feature will be very useful if more than one developer is working on the same file in the same time and will prevent overlapping changes and files inconsistency. [40]

3.2.2 Objects Identification Mechanisms

The identification of an object in RCS system was established through markers, which was special strings placed in the code of a revision “for instance inside a comment”, and then those markers are to be replaced by the RCS by the following information:

$Id: filename, revision-number, date, time, author, state, locker $

This string is auto generated, so the developer need not change it during the development lifecycle as the check-out command will keep it up to date.

The command “ident” extracts such markers from any file, in particular from object code. “Ident” helps to find out which revisions of which modules were used in a given program. It returns a complete and unambiguous component list, from which a copy of the program can be reconstructed. This facility is invaluable for program maintenance. [30]

The command “Log” accumulates log messages that are requested during the check-in operations on the file, this helps the developer maintain a full list of what was changed in each revision and by whom.

And because of the space limitations back in the 1980s, the log was physically stored only once, and other versions were stored by differences only to save the storage space from depletion. This mechanism is known as Deltas and will be discussed later in this chapter.

3.2.3 Objects tree structure

The starting point of an object creation in RCS is the check-in command which creates a revision tree with the root having the version number of 1.1 (unless explicitly defined otherwise) and each new revision increments the revision numbering by 0.1 (unless explicitly defined otherwise).

The first digit of the version number is called the release number and the second digit is the level number, usually when an object is changed, only the level number increases, unless there was a major change in that object and hence the release number must be changed too. [26]

A young revision tree is straightforward tree which means it has no side branches, as seen in figure 3.1 side branches may start appearing when the tree ages, and those branches are needed for the following reasons:

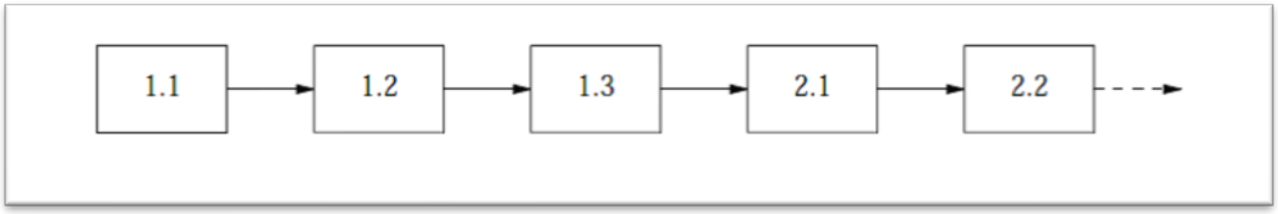


Figure 3.1: Straightforward young revision tree [30]

Temporary fixes

RCS does not allow extra revisions in between already defined two revisions, and thus if a revision tree has moved on to have multiple revisions past an old revision, and new requirements arises for an already deployed old revision "let's suppose that these new requirements arose for version 1.3 in figure 3.1" and since the RCS prohibits the addition of in-between versions, the need for a side branch arise.

The new branch will have the version number of 1.3.1.1, the double numbering here allows the addition of extra new branches on the same source node (1.3), so another branch will have the version of 1.3.2.1, and the new tree structure is illustrated in figure 3.2.

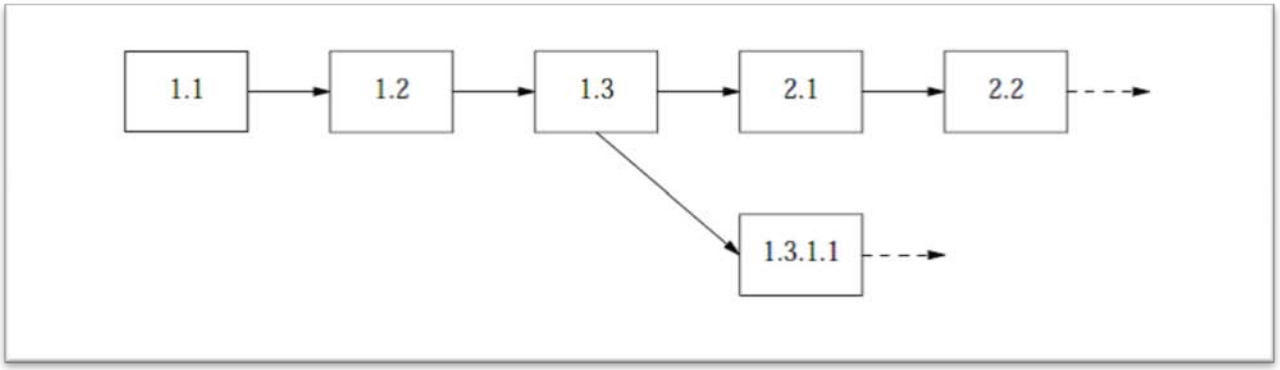


Figure 3.2: Branched RCS revision tree [30]

Distributed development and customer modifications

Each customer can have his own tree of CVS revisions, and because the customers usually does not have all revisions implemented in their site, the CVS revision tree implemented there can have only limited versions and not all developed revisions.

In this case each customer revision will be placed in a side branch so local modifications can be derived from each received version for each customer.

Parallel development

Sometimes it is desirable to explore an alternate design or a different implementation technique in parallel with the main line development. Such development should be carried out on a side branch. The experimental changes may later be moved into the main line, or abandoned. [30]

Conflicting updates

Two developers or more might need to check out the same CI at the same time, or when a developer checks out a file and leaves it that way for later modifications, and other developers need to also check out the same file. In that case, the file would be checked out and modified, and since the second developer cannot check-in the file again to the repository, the developer creates a side branch and waits for the first developer to check-in his file again and then resolve the conflicts and merge the branch back to the main branch.

Every node in a revision tree consists of the following attributes: a revision number, a check-in date and time, the author's identification, a log entry, a state and the actual text. All these attributes are determined at the time the revision is checked in. The state attribute indicates the status of a revision. It is set automatically to 'experimental' during check-in. A revision can later be promoted to a higher status, for example 'stable' or 'released'. The set of states is user-defined. [30]

3.2.4 Revisions Differentiated Through Deltas

When RCS was first introduced in the 1980s, the storage capabilities were extremely low. Hence RCS used Deltas for conserving space, RCS stores revisions in the form of deltas, which are the differences between revisions. The user interface completely hides this fact.

Instead of keeping all copies of a modified file, Delta keeps record of changes to the file and can build that file from its original state using those change records. A delta is a serial sequence of edit commands that transforms one string into another. The deltas employed by RCS are line-based, which means that the only edit commands allowed are insertion and deletion of lines. If a single character in a line is changed, the edit scripts consider the entire line changed. The

program diff2 produces a small, line-based delta between pairs of text files. A character-based edit script would take much longer to compute, and would not be significantly shorter. [17]

Deltas are a series of computational activities; hence it increases the access time to a file especially if it was changed a lot. If access time was high, it would greatly discourage developers and prevent them from gaining maximum productivity from the RCS system. But in the same time, deltas offer more storage space through storing only differences, so it is a basic time-space tradeoff to consider depending on the project's priorities.

RCS stores a copy of the latest version of the CI, and stores only deltas for previous versions. This way the access time would improve noticeably because this algorithm has the advantage that extraction of the latest revision is a simple and fast copy operation. Also it uses reverse deltas to gain access to previous CIs using the latest copy of the CI and the changes that occurred to the previous states of that CI.

Branches are not treated the same as basic CIs. The simplest solution would be to store complete copies for the tips of all branches. Obviously, this approach would cost too much space. Instead, RCS uses forward deltas for branches. Regenerating a revision on a side branch proceeds as follows. First, extract the latest revision on the trunk; secondly, apply reverse deltas until the fork revision for the branch is obtained; thirdly, apply forward deltas until the desired branch revision is reached. [30] Figure 3.3 illustrates a tree with one side branch.

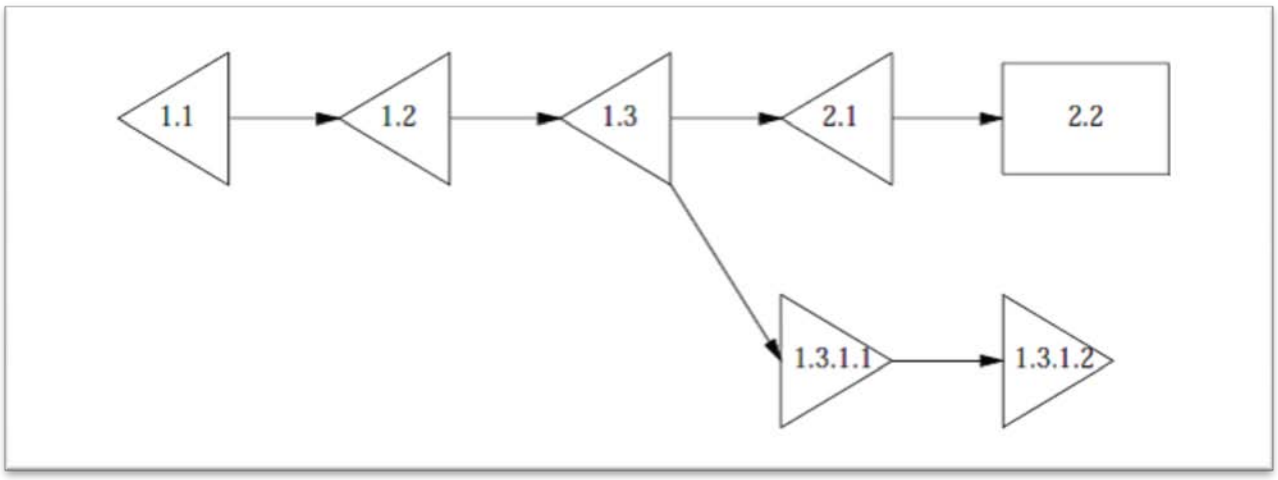


Fig 3.3: Branches representations in RCS [30]

3.2.5 RCS Parallel Access

RCS uses exclusive locks mechanism on files that are checked out of the project's repository for editing by some developer. RCS must prevent two or more persons from depositing competing changes of the same revision.

Suppose two programmers check out revision 2.4 and modify it. Programmer A checks in a revision before programmer B. Unfortunately, programmer B has not seen A's changes, so the effect is that A's changes are covered up by B's deposit. A's changes are not lost since all revisions are saved, but they are confined to a single revision. This situation is prevented in RCS by locking. Whenever a developer intends to edit a revision, the revision should be checked out and locked, using the −l option on co. On subsequent check-in, ci tests the lock and then removes it. In exclusive locks algorithms at most one programmer at a time may lock a particular revision, and only this programmer may check in the succeeding revision. Thus, while a revision is locked, it is the exclusive responsibility of the locker. [30]

A dilemma here is that RCS must not stand in the way of making progress with a project. Hence the exclusive-locking mechanism had to find workarounds for working in parallel. In RCS even if a file is checked-out, other developers can still gain a copy of that file for observation or building purposes keeping in mind that they cannot lock an already locked file.

Also they can check-in an already locked file on another branch, but if a lock has been on a file for a very long time, the administrator can break the lock on that file and release it to other developers. In this case an email notification would be sent to the owner of the CI locked.

There is an alternative way of operation in RCS file management, which is private files, and in this case when a programmer owns an RCS file and does not expect anyone else to perform check-in operations, the owner will be the only person who can modify the file, and thus the locking mechanisms become unnecessary. In this case, the 'strict locking feature' may be disabled, provided that file protection is set such that only the owner may write the RCS file. This has the effect that only the owner can check-in revisions, and that no lock is needed for doing so.

3.2.6 Limitations of RCS:

RCS system is a bit trivial when compared to current versioning systems, but then it was very popular and provided the base for other versioning systems to build on it and to learn from its mistakes. Listed below are some obvious and main limitations of RCS [41]:

1- Lacks GUI interface that could be a user-friendly alternative for the command line interface which now is outdated.
2- It uses the lock-modify-commit mechanism so it does not allow more than one developer to modify the same file as it will be locked, it will only retrieve a working copy but will not commit his changes to the main repository, only the developer who checked out the file can check it back again into the project’s repository.
3- The lock-modify-commit mechanism can be useful for maintaining websites of the web development nature which can be slowly changing and does not have to deal with conflicts between simultaneous edits from several developers.

4- The developer cannot see a read-only version of the file unless he checks out the file and a record for that check-out is created, and that could be considered an overhead especially if that developer wants only to review the file and not to change it.
5- Up-to-date warnings have to be sent to all members who have checked-out a certain file, if that file has been changed by some other developer, and this mechanism is carried out in RCS only by the developers manual update command.

3.3 Concurrent Versions System

In April, 1989, Brian Berliner designed and coded CVS. CVS is the main successor for the RCS revision control system, CVS (Concurrent Versions System) is a front end to the RCS revision control system which extends the notion of revision control from a collection of files in a single directory to a hierarchical collection of directories each containing revision controlled files. Directories and files in the CVS system can be combined together in many ways to form a software release. CVS provides the functions necessary to manage these software releases and to control the concurrent editing of source files among multiple software developers. [5]

Almost every developer will have used CVS at some time because of its strong presence on the software development scene for at least two decades. CVS is a version control system. Using it, the user can record the history of his source files. For example, bugs sometimes creep in when software is modified, and you might not detect the bug until a long time after you make the modification. With CVS, you can easily retrieve old versions to see exactly which change caused the bug. This can sometimes be a big help. [46]

Record keeping became necessary because people wanted to compare a program's current state with how it was at some point in the past. For example, in the normal course of implementing a new feature, a developer may bring the program into a thoroughly broken state, where it will probably remain until the feature is mostly finished. Unfortunately, this is just the time when someone usually calls to report a bug in the last publicly released version. To debug the problem (which may also exist in the current version of the sources), the program has to be brought back to a useable state. [46]

For space conserving purposes, CVS stores deltas for changes files and do not keep the whole copies. CVS also enables team members to coordinate their work in the project. RCS used the exclusive lock technique to avoid conflicts between the developers. But in CVS developers can check out any file even if it has been checked out before by someone else. The main technique is to isolate each developer’s workspace from other developers, meaning that each developer has

his own copy of the files that he's modifying. Then when time comes for check-ins, CVS handles the conflicts resolution between each developer's updated versions.

3.3.1 New Concepts in CVS

CVS introduced new concepts to versioning in a number of ways: it used a client/server model, supported branch imports, used unreserved checkouts and utilized symbolic mapping. Concepts are listed below along with further discussion [26]:

3.3.1.1 Client/Server Model

This enabled developers at disparate locations to function as a team even when encumbered by slow modems as it stored the version history on a central server whilst the developers worked on a copy of the files on their client machines. However, this meant that network access was required when CVS operations, such as updates or checkins, needed to be performed.

3.3.1.2 Branch Imports

In cases where developers needed to maintain their own version of files, CVS allowed one team to import branches from another and merge the changes, if required.

3.3.1.3 Unreserved Checkouts

Unreserved checkouts allow more than one developer to work on the same files at the same time so that there is no lock on files when they are checked out.

3.3.1.4 Symbolic Mapping

CVS has a database that provides symbolic mapping of names to components of a larger software distribution, thus facilitating the naming of collections of directories and files. The result is that a single command can manipulate an entire collection of directories and files.

3.3.2 Terminologies

To start reviewing the CVS tool, we need to clarify the terms that are used in CVS, and hereunder is a list of the most relevant terms and its clarification [3]:

- Check out: To request a working copy from the repository. Your working copy reflects the state of the project as of the moment you checked it out; when you and other developers make changes, you must use the commit and update commands to "publish" your changes and view others' changes.
- Commit: To send changes from your working copy into the central repository. Also known as check in.

- Conflict: The situation when two developers try to commit changes to the same region of the same file. CVS notices and points out conflicts, but the developers must resolve them.
- Log message: A comment you attach to a revision when you commit it, describing the changes. Others can page through the log messages to get a summary of what's been going on in a project.
- Repository: The master copy where CVS stores a project's full revision history. Each project has exactly one repository.
- Revision: A committed change in the history of a file or set of files. A revision is one "snapshot" in a constantly changing project.
- Update: To bring others' changes from the repository into your working copy and to show whether your working copy has any uncommitted changes.
- Working copy: The copy in which you actually make changes to a project. There can be many working copies of a given project; generally, each developer has his or her own copy.

3.3.3 Versioning Mechanism

Each version of a file has a unique revision number. Revision numbers look like `1.1', `1.2', `1.3.2.2' or even `1.3.2.2.4.5'. A revision number always has an even number of period-separated decimal integers. By default revision 1.1 is the first revision of a file. Each successive revision is given a new number by increasing the rightmost number by one.

CVS enables the development team to create new branches of code, separating them from the main codeline when necessary. And also the owner of the branch can merge back to the main codeline.

Each branch has a branch number, consisting of an odd number of period-separated decimal integers. The branch number is created by appending an integer to the revision number where the corresponding branch forked off. Having branch numbers allows more than one branch to be forked off from a certain revision. [46]

All revisions on a branch have revision numbers formed by appending an ordinal number to the branch number. When CVS creates a branch number it picks the first unused even integer, starting with 2. So when you want to create a branch from revision 6.4 it will be numbered 6.4.2. All branch numbers ending in a zero (such as 6.4.0) are used internally by CVS. [46]

3.3.4 Parallel Development in CVS

A well-know methodology in controlling parallel development is the lock-modify-unlock development model, which is used by RCS as discussed in the previous section, wherein a

developer first obtains exclusive write access (a lock) to the file to be edited, makes the changes, and then releases the lock to allow other developers access to the file. If someone else already has a lock on the file, they have to "release" it before others can lock it and start making changes.

This model is workable if the developers know each other, know who's planning to do what at any given time, and can communicate with each other quickly in case someone cannot work because of access conflict. However, if the developer group becomes too large or too spread out, dealing with all the locking issues begins to be more complex at coding time. It quickly becomes a constant hassle that can discourage people from getting real work done. [2]

CVS allows several software developers to edit personal copies of a revision controlled file concurrently. The revision number of each checked out file is maintained independently for each user, and CVS forces the checked out file to be current with the "head" revision before it can be "committed" as a permanent change. A checked out file is brought up-to-date with the "head" revision using the "update" command of CVS. This command compares the "head" revision number with that of the user's file and performs an RCS merge operation if they are not the same. The result of the merge is a file that contains the user's modifications and those modifications that were "committed" after the user checked out his version of the file (as well as a backup copy of the user's original file). CVS points out any conflicts during the merge. It is the user's responsibility to resolve these conflicts and to "commit" his/her changes when ready. [5]

CVS allows developers to edit simultaneously, and takes the burden of integrating all the changes, and keeps track of any conflicts. This process uses the copy-modify-merge model, which works as follows [2]:

1. The developer downloads a working copy (a directory tree containing the files that make up the project) from CVS. This is also known as "checking out" a working copy, like checking a book out of the library.

2. The developer edits freely in his working copy. At the same time, other developers might be busy in their own working copies. Because these are all separate copies, there is no interference. It is as if all of the developers have their own copy of the same library book, and they're all at work scribbling comments in the margins or rewriting certain pages independently.

3. The developer finishes his changes and commits them into CVS along with a "log message," which is a comment explaining the nature and purpose of the changes. This is like informing the library of what changes he made to the book and why. The library then incorporates these changes into a "master" copy, where they are recorded for all time.

4. Meanwhile, other developers can ask CVS to query the library to see if the master copy has changed recently. If it has, CVS automatically updates their working copies. (This part is magical and wonderful, and we hope you appreciate it. Imagine how different the world would be if real books worked this way!)

CVS does not force resynchronizations between a developer's workspace and the project's repository and does consider all developers on a project are equal. Deciding when to update or when to commit is largely a matter of personal preference or project policy. One common strategy for coding projects is to always update before commencing work on a major change and to commit only when the changes are complete and tested so that the master copy is always in a stable state. [23]

3.3.5 CVS Repository

The CVS repository is simply a file tree kept on a central server that is installed on a UNIX OS. The repository is a main component in the Version Control System, as it stores all CIs and its corresponding versions and tags, alongside other useful information that is used internally by the system. To understand what CVS repository provided functionalities are, consider the following scenario [2]:

1. Two developers, Joseph and Sarah, check out working copies of a project at the same time. The project is at its starting point-no changes have been committed by anyone yet, so all the files are in their original, pristine state.

2. Sarah gets right to work and soon commits her first batch of changes.

3. Meanwhile, Joseph is not doing any work at all.

4. Sarah, feeling quite productive, commits her second batch of changes. Now, the repository's history contains the original files, followed by Sarah's first batch of changes, followed by this set of changes.

5. Meanwhile, developer Joseph remains inactive.

6. Suddenly, developer Robert joins the project and checks out a working copy from the repository. Robert's copy reflects Sarah's first two sets of changes, because they are already in the repository when Robert checks out his copy.

7. Sarah completes and commits her third batch of changes.

8. Finally, unaware of the recent frenzy of activity, Joseph decides it's time to start work. He doesn't bother to update his copy; he just commences editing files, some of which might be files that Sarah has worked in. Shortly thereafter, Joseph commits his first changes.

At this point, one of two things can happen. If none of the files Joseph edited have been edited by Sarah, the commit succeeds. However, if CVS detects that some of Joseph's files are out of date with respect to the repository's latest copies, and those files have also been changed by Joseph in his working copy, CVS informs Joseph that he must do an update before committing those files. [2]

When Joseph runs the update, CVS merges all of Sarah's changes into Joseph's local copies of the files. Some of Sarah's work might conflict with Joseph's uncommitted changes, and some might not. Those parts that don't are simply applied to Sarah's copies without further complication; Joseph must resolve the conflicting ones before they can be committed. [2]

If developer Robert does an update now, he will receive several batches of changes from the repository: Sarah's third commit, then Joseph's first, and then possibly Joseph's second commit (if Joseph had to resolve any conflicts). [2]

However, CVS repository is mainly designed for archiving data. CVS offers only a basic querying interface for retrieving a given version of a file or an attribute list with the file state evolution. CVS provides no features to let users get data overviews easily. Another challenge of CVS repository is the data size and complexity. Raw repository information is too large and provides directly just limited insight in the evolution of a software project. Extra analysis is needed to process these data and extract relevant evolution features. [33]

3.3.6 CIs Differences Stored as Diffs

In order for CVS to serve up changes in the correct sequence to developers whose working copies might be out of sync by varying degrees, the repository needs to store all commits since the project's beginning. In practice, the CVS repository stores them all as successive diffs. Consequently, even for a very old working copy, CVS is able to calculate the difference between the working copy's files and the current state of the repository, and is thereby able to bring the working copy up to date efficiently. This makes it easy for developers to view the project's history at any point and to revive even very old working copies. [46]

This demonstrates that CVS stores only the differences between revisions, thereby saving a lot of space compared with storing each revision in full. To go backwards from the most recent revision to the previous one, it patches the latter revision using the stored diff. Of course, this means that the further back you go in the revision history, the more patch operations must be performed. (For example, if the file is on revision 1.7 and CVS is asked to retrieve revision 1.4, it has to produce 1.6 by patching backwards from 1.7, then 1.5 by patching 1.6, then 1.4 by patching 1.5.) Fortunately, old revisions are also the ones least often retrieved, so the RCS system performance is good in practice. [5]

Although, strictly speaking, the repository could achieve the same results by other means, in practice, storing diffs is a simple, intuitive means of implementing the necessary functionality.

The process has other added benefits. By using patch appropriately, CVS can reconstruct any previous state of the file tree and thus bring any working copy from one state to another. It can allow someone to check out the project as it looked at any particular time. It can also show the

differences, in diff format; between two states of the tree without affecting anyone's working copy.

As a result, the very features necessary to give convenient access to a project's history are also useful for allowing a group of decentralized developers working separately to collaborate on the project.

3.3.7 Limitations of CVS

CVS as the successor of RCS had more functionalities and learned from some of the mistakes that were made in RCS. But of course, the concept of VCS and SCM was still fresh back then when CVS was released, and CVS still had missing features and drawbacks. Listed below are the most obvious limitations [19]:

1. User interface is clumsy; Administration and use require much skills
2. CVS is notoriously inflexible when trying to modify an existing repository's structure. It does not allow files to be moved, copied, or renamed without splitting the file's history (so that you need to know its old name to see its old history).
3. CVS does not allow directories to be moved around (or even deleted) without editing the repository directly
4. The developer cannot see a read-only version of the file unless he checks out the file and a record for that check-out is created, and that could be considered an overhead especially if that developer wants only to review the file and not to change it.
5. No tools to check the health of the repository
6. CVS occasionally crashes, leaving dangling file locks and hung processes
7. Unable to detect logical conflicts between files
8. A network-accessible CVS repository must be installed on a UNIX box, if installed on other operating systems platforms; it will be accessible only locally on that machine.

3.4 Subversion

In software development, Apache Subversion (SVN) is a version control system founded and sponsored in 2000 by CollabNet Inc. Developers use Subversion to maintain current and historical versions of files such as source code, web pages, and documentation. Its goal is to be a mostly-compatible successor to the widely used Concurrent Versions System (CVS). [49]

In early 2000, CollabNet, Inc. began seeking developers to write a replacement for CVS. CollabNet offers a collaboration software suite called CollabNet Enterprise Edition (CEE), of which one component is version control. Although CEE used CVS as its initial version control

system, CVS's limitations were obvious from the beginning, and CollabNet knew it would eventually have to find something better. Unfortunately, CVS had become the typical standard in the open source world largely because there wasn't anything better, at least not under a free license. So CollabNet determined to write a new version control system from scratch, retaining the basic ideas of CVS, but without the bugs and misfeatures. [26]

In February 2000, they contacted Karl Fogel, the author of Open Source Development with CVS (Coriolis, 1999), and asked if he would like to work on this new project. Coincidentally, at the time Karl was already discussing a design for a new version control system with his friend Jim Blandy. In 1995, the two had started Cyclic Software, a company providing CVS support contracts, and although they later sold the business, they still used CVS every day at their jobs. Their frustration with CVS had led Jim to think carefully about better ways to manage versioned data, and he'd already come up with not only the name "Subversion," but also the basic design of the Subversion data store. [26]

Subversion as an open-source version control system is meant to succeed the CVS, the main points that developers of subversion focused on are:

- Enhancing the repository structure and reliability, plus offering two types of repositories for subversion implementation.
- No local working copies in each developer’s local machine for active files only. Which means that unless the developer wants an older version of some CI, he can connect directly to the repository server and update the latest versions, without making local copies to workspaces.
- Enhanced updates conflict resolution algorithms, through displaying three different types of the conflicted CI to assist in resolving the conflict.

Subversion manages files and directories, and the changes made to them, over time. This allows the user to recover older versions of your data or examine the history of how your data changed. Subversion can operate across networks, which allows it to be used by people on different computers. At some level, the ability of various people to modify and manage the same set of data from their respective locations encourages collaboration. Progress can occur more quickly without a single conduit through which all modifications must occur.

3.4.1 Subversion Basics

The basic interface to Subversion is very similar to CVS. The primary method of interfacing with Subversion is through the command line (although there are some very good GUI front ends available), and by design, SVN has very similar command-line operations to CVS.

Most of the CVS Commands have not been renamed, and for the most part, if a user is familiar with how to do something in CVS, doing the same thing in SVN will have similar syntax.

Subversion has a larger feature-set; Subversion has many commands that don't have a CVS counterpart. The concepts of subversion are similar to many other version control systems, and the differences should be easy to learn. [19]

Subversion uses a client-server architecture, where a central repository is installed on a server and clients check out local working copies where they can modify CIs. After the CI reaches an acceptable state of modifications, the owner of that CI resynchronizes his local workspace to the server's repository and makes his updates available to other developers.

3.4.2 Repository Structure

Subversion is a centralized system for sharing information. At its core is a repository, which is a central store of data. The repository stores information in the form of a filesystem tree, a typical hierarchy of files and directories. Any number of clients can connect to the repository, and then read or write to these files. By writing data, a client makes the information available to others; by reading data, the client receives information from others. [9]

Basically Subversion's repository is a regular folder structure residing in an operating system storage space and making use of its embedded security model. But the difference is that each change to any of the repository assets would be tracked and recorded in a record database.

Subversion allows for flexibility in layout of repositories, by keeping revision histories for both files and directories across moves, copies, and renames. This can become helpful in providing flexibility in changing the repository structure.

CVS is inflexible when trying to modify an existing repository's structure. It does not allow files to be moved, copied, or renamed without splitting the file's history (so that you need to know its old name to see its old history). Also, CVS doesn't allow directories to be moved around (or even deleted) without editing the repository directly. Similarly, moves, renames, and copies are difficult in Microsoft's Visual SourceSafe (which will be presented later in this chapter), although not quite as hard as CVS. On the other hand, with Subversion, you can move, rename, and copy files and directories easily without any worry that you will lose (or split) your history or corrupt older revisions, and that's all because of the flexible repository design of the Subversion system. [19]

Subversion has a repository backend that allows different types of repository storage systems to be plugged in, transparently to the client. Originally, the only actual repository storage system that was available was the Berkeley DB database system. As of release 1.1 of Subversion, though, a file system based backend (FSFS) is also available as part of the core Subversion system. Instead of storing the entire repository database in a single monolithic database, like the Berkeley DB backend does, the Subversion FSFS storage uses individual files for each revision in the repository. So, when you commit revision 3529, there will be a file named 3529 created,

which holds all of the changes for that revision, regardless of how many versioned files were changed in that revision. [19]

By offering two types of repositories to the implementers of the version system, subversion gained advantage over similar systems and provided more flexibility.

In comparison between the two repository modes for Subversion, the next points clear the obvious advantages for FSFS over BDB [39]:

- Write access not required for read operations

To perform a checkout, update, or similar operation on an FSFS repository requires no write access to any part of the repository.

- Smaller repositories

An FSFS repository is smaller than a BDB repository. Generally, the space savings are on the order of 10-20%, but if you do a lot of work on branches, the savings could be much higher, due to the way FSFS stores deltas. Also, if you have many small repositories, the overhead of FSFS is much smaller than the overhead of the BDB implementation.

- Platform-independent

The format of an FSFS repository is platform-independent, whereas a BDB repository will generally require recovery (or a dump and load) before it can be accessed with a different operating system, hardware platform, or BDB version.

- Can be hosted on network filesystem

FSFS repositories can be hosted on network filesystems, just as CVS repositories can.

- Standard backup software

An FSFS repository can be backed up with standard backup software. Since old revision files do not change, incremental backups with standard backup software are efficient.

- Can split up repository across multiple spools

If an FSFS repository is outgrowing the filesystem it lives on, you can symlink old revisions off to another filesystem.

- More easily understood repository layout

If something goes wrong and you need to examine your repository, it may be easier to do so with the FSFS format than with the BDB format.

3.4.3 CI Versioning

Subversion versions each CI contained in its repository to uniquely identify it, thus making it easy to retrieve a certain set of CIs and updating them. Subversion allows developers to work directly on the repository but only on the latest versions of CIs (active CIs), and requires the developers to make formal check-outs when retrieving older versions (inactive CIs) to update them.

Subversion uses a network protocol named WebDAV to transfer CIs between working spaces and the main repository. Using the WebDAV protocol, the user can mount the subversion repository into his local machine and transfer files to and from the repository. [8]

The core mission of a version control system is to enable collaborative editing and sharing of data. But different systems use different strategies to achieve this. It is important to understand these different strategies, for a couple of reasons. First, it will help you compare and contrast existing version control systems, in case you encounter other systems similar to Subversion. Beyond that, it will also help you make more effective use of Subversion, since Subversion itself supports a couple of different ways of working. So developers would be accessing the latest versions of CIs without having to update their workspaces often to see other developers' updates.

Subversion's CI Versioning mechanisms does have its downsides. For one, it requires a reasonably fast network connection to the computer that's serving the repository, so it may not be practical for a laptop that's frequently used away from home; although if you have access to a network connection back to the server, it is always possible to copy files to your local hard drive, edit them and then copy them back to the repository. Another downside to CI Versioning is that you can access only the most recent repository revision. If you want to access older revisions of files, you have to download them locally, which can be done by checking out a directory at a specific revision. [28]

3.4.4 Parallel CIs Access

Version control systems in general tend to face the same problem when it comes to collaborating team development. The VCS should be able to control access to shared CIs either by using the lock-modify-unlock paradigm, or by using the copy-modify-merge paradigm to prevent users from overwriting each other's work accidently.

Consider this scenario, Suppose we have two developers, Harry and Sally. They each decide to edit the same repository file at the same time. If Harry saves his changes to the repository first, it's possible that (a few moments later) Sally could accidentally overwrite them with her own new version of the file. While Harry's version of the file won't be lost forever (because the system remembers every change), any changes Harry made won't be present in Sally's newer

version of the file, because she never saw Harry's changes to begin with. Harry's work is still effectively missing from the latest version of the file-and probably by accident. [9]

Many version control systems use a lock-modify-unlock model to address the problem of many authors overwriting each other's work. In this model, the repository allows only one person to change a file at a time. This exclusivity policy is managed using locks.

Subversion does not encourage the exclusive locks mechanism but does provide it because in some situations it is appropriate to lock files in the central repository, consider the example below:

“The copy-modify-merge model is based on the assumption that files are contextually mergeable that is, the majority of the files in the repository are line-based text files (such as program source code). But for files with binary formats, such as Fundamental Concepts artwork or sound, it is often impossible to merge conflicting changes. In these situations, it really is necessary for users to take strict turns when changing the file. Without serialized access, somebody ends up wasting time on changes that are ultimately discarded.” [9]

There are several drawbacks of the lock-modify-unlock paradigm, which are that Locking may cause administrative problems when Sometimes a developer will lock a file and then forget about it which will cause a lot of unnecessary delay and wasted time until the administrator resolves the situation, and Locking may cause unnecessary serialization like when two developers want to edit different parts of a file, hence there's no need for them to take turns in this situation, and also Locking may create a false sense of security in cases when two developers lock different but semantically connected files, and then change them and both of them became unstable when assembled together in the same build.

Subversion mainly uses a copy-modify-merge model as an alternative to locking. In this paradigm, each developer’s client connects to the project’s repository and creates a personal working copy which is a local reflection of the repository's files and directories. Users then work simultaneously and independently, modifying their private copies. Finally, the private copies are merged together into a new, final version. The version control system often assists with the merging, but ultimately, a human being is responsible for making it happen correctly.

To illustrate the copy-modify-merge model, consider the example where two developers say Harry and Sally each create working copies of the same project, copied from the repository. They work concurrently and make changes to the same file A within their copies. Sally saves her changes to the repository first. When Harry attempts to save his changes later, the repository informs him that his file A is out of date. In other words, file A in the repository has somehow changed since he last copied it. So Harry asks his client to merge any new changes from the repository into his working copy of file A. Chances are that Sally's changes don't overlap with his own; once he has both sets of changes integrated, he saves his working copy back to the repository. [9]

3.4.5 Conflict Resolution

Subversion uses a prototype of making modifications and then merging them with the modifications others have made to resolve update conflicts as they occur in the project.

Subversion solves conflicts in a pair of ways. First, when a conflict occurs in a file, Subversion records the fact that the file is in a state of conflict, and won't allow the developer to commit changes to that file until he explicitly resolves the conflict. Second, Subversion provides interactive conflict resolution, which allows the developer to resolve conflicts as they happen instead of having to go back and do so after the update or merge operation completes. [9]

If a conflict occurs, SVN replaces the file with one containing diffs, however, it also adds temporary versions of the file with the local version, the server version, and the local version prior to any changes. These extra files make resolving conflicts clearer, and once the conflict has been settled, a call to “svn resolved” removes the extra files and the conflict indicator on the file.

3.4.6 Branching and Tagging

Subversion uses the inter-file branching model to handle branches and tags. A branch is a copy of the main codeline that satisfy a certain set of requirements different from the original codeline and does evolve rapidly and becomes more and more different from its original source.

Tagging a codeline is a copy of a codeline like the branch, but is meant to freeze a snapshot of the codeline and would not differ from the original state of the original codeline.

The system sets up a new branch or tag by using the 'svn copy' command, which should be used instead of the native operating system mechanism. Subversion does not create an entire new file version in the repository with its copy. Instead, the old and new versions are linked together internally and the history is preserved for both. The copied versions take up only a little extra room in the repository because Subversion saves only the differences from the original versions. [49]

All the versions in each branch maintain the history of the file up to the point of the copy, plus any changes made since. One can "merge" changes back into the trunk or between branches. To Subversion, the only difference between tags and branches is that changes should not be checked into the tagged versions. Due to the differencing algorithm, creating a tag or a branch takes very little additional space in the repository. [49]

In figure 3.4, the illustration of a typical software development project is shown with its tags, branches and trunks.

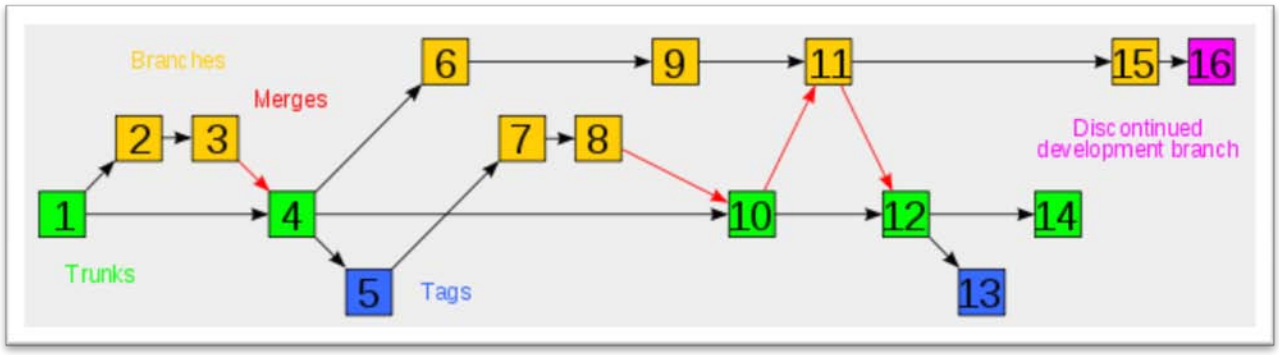


Fig 3.4: Typical Subversion software development project [49]

3.4.7 Limitations of Subversion

Subversion has gained popularity in the field of version control since its release, but many shortages were found in its implementation. Some of them are tackled by next releases of the subversion system, and some are still there.

The following features are some that are either missing from Subversion, or severely limited in their implementation [43]:

- **Locking**

Subversion currently has no support for file locking, the file locking is supposed to prevent more than one person from working on a particular file at a given point in time. Locking can be useful when dealing with binary files that are not easily merged. According to the Subversion developers, however, locking is on the list of features to be implemented in the next few releases of the system.

- **Distributed Repository**

Subversion does not currently have any support for distributed repositories. Distributed repositories can be extremely important for some large projects, especially open source projects.

- **Merging History**

When Subversion merges between two branches, it merges all of the differences between the first location and the second location into a third location in the working copy of the repository. This means that if a developer wants to merge changes made on a branch back into the main trunk (or to another branch), the developer needs to specify the differences in revisions on the branch, where the changes to be merged occurred. Unfortunately, this means that the user must keep the accounting information describing when the branch was created (and when it was last merged) in order to ensure that the correct data is merged.

- **WebDAV File Transfer**

Subversion uses the WebDAV file transfer protocol to exchange files between working copies and the central repository, many operating systems can mount the repository as regular directory structure through which a developer can gain direct access to the repository without referring to the subversion control rules which can cause chaos.

3.5 Perforce

Perforce is a Software Configuration Management (SCM) system based on client/server architecture. Users of Perforce client programs connect to a Perforce server and use Perforce client programs to transfer files between the server's file repository and individual client workstations as described in Figure 3.5 below.

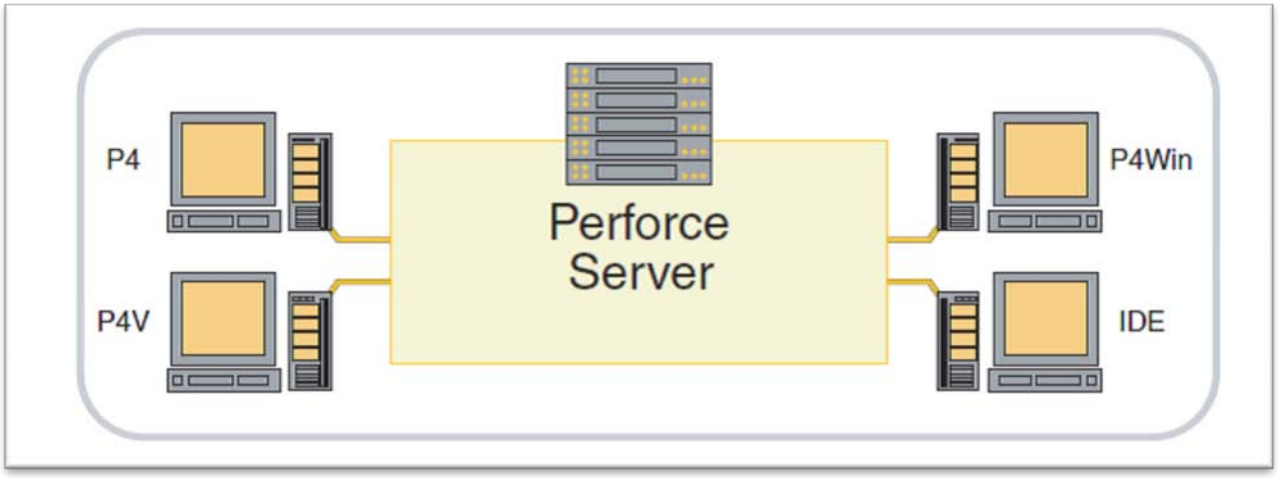


Fig 3.5: Files exchange between Perforce server and clients [22]

The perforce server manages a central database and a master repository of file versions. Users work on files in local client workspaces, and submit changed files together in using changelists. Communication from client to the server is via any of a number of clients (user interfaces). Client and server communicate via TCP/IP using a proprietary RPC and streaming protocol. [42]

The Perforce server manages the master file repository, or depot. There can be more than one depot per server. The depots contain every revision of every file under Perforce control. Perforce organizes files in depots into directory trees, like a large hard drive. Files in a depot are referred to as depot files or versioned files. The server maintains a database to track change logs, user

permissions, and which users have which files checked out at any time. The information stored in this database is referred to as metadata. [22]

Perforce servers use native operating system capabilities to manage the database and the versioned files, and require no dedicated filesystems or volumes.

When using Perforce as the configuration management system in a software development company, the developer never interacts directly with files in the depot. Instead, he uses a Perforce client program to manage a specially-designated area of his local workstation called a client workspace. A client workspace contains a local copy of a portion of the depot.

For developers to be able to connect to the perforce server and gain access to its services and databases (repository) client applications and its interfaces must be installed on the clients' machines. Perforce client programs enable the developer to check files in and out, manage conflicts, create development branches, track bugs and change requests. Perforce client programs include [22]:

- P4, the Perforce Command-Line Client, for all platforms
- P4V, the Perforce Visual Client, for Mac OS X, UNIX, Linux, and Windows
- P4Web, the Perforce Web Client, a browser-based client
- Integrations, or plug-ins, that work with commercial IDEs and productivity software

Perforce is a client\server SCM tool that provides change management capabilities using changelists and defect tracking techniques (which will be discussed later), alongside the common functionalities of version control. Perforce differs from version control systems introduced earlier in this chapter, in its capability to govern the process of change in an automated fashion.

To use Perforce, the administrator must set up the developers' Perforce client program to connect to the organization's Perforce server, and then the administrator should guide the client program where his client workspace is located on the local hard drive, also a set of files to be retrieved and a set of permissions must be defined for each user to be able to access resource on the Perforce server and to start his development tasks.

The perforce server grants copies to each developer's workspace on his local machine using the installed client program. This would require less network traffic, and hence client programs can work offline and does not require a constant connection to the perforce server.

Perforce client program communicates with Perforce servers over a TCP/IP connection. Each Perforce client program needs to know the address and port of the Perforce server with which it communicates. The address and port are stored in the P4PORT environment variable; depending on the Perforce client the user is using. [22]

3.5.1 Changelists and Atomic Transactions

Changelists are a compilation of CIs transactions that is used by SCM tools to coordinate the editing, adding and deleting CIs from the SCM repository. The developer extracts the needed files from the repository onto his local workspace, then uses the client program of perforce to create a changelist that keeps records of what files are intended for changes.

The changelist records the list of CIs and its respective versions, alongside information about the changes that were made to those CIs, and each changelist has its own unique identifier generated by perforce. The information that corresponds with the changelist is provided by the change implementer as guidance for other developers regarding the change in hand.

The changelists are atomic transactions by nature, which means that the perforce server would either accept the whole changes proposed in the changelist, or ignore all those changes. By submitting a changelist, the developer who created it has to deal with conflicts, and if he was unable to resolve conflicts between his updates and other developer's updates, then he has to either ignore the whole changelist, or leave it as pending changelist until conflicts are fully resolved.

In the Perforce model, multi-file changes can be treated as atomic transactions. Database integrity is assured at each server transaction, and each logical change is uniquely and permanently identified. The inherent data aggregation of atomic changes provides an implicit relationship between file versions and external representations of work such as bug fix orders, or program specifications. Perforce tracks additions, deletes and renames of files as well as updates. Perforce allows users to selectively synchronize their client workspaces with submitted changes as desired. [44]

3.5.2 Working Concurrently

Perforce offers the ability of shared access to the same repository CI, in which teams can work concurrently. But of course in this case the drawback is having to deal with update conflicts between parallel check-ins. The conflict resolution and three-way merge process enables multiple users to work on the same files at the same time effectively and without replacing each other's work by mistake.

Perforce supports two types of file locking. You can prevent files from being checked in with file locking and you can prevent file checkout with exclusive-open [22]:

- To prevent other users from checking in changes to a file the developer is working on, he locks the file. Other users can still check out your locked file, but are unable to submit changelists that affect your locked file until you submit your changes.
- The developer can prevent a file from being checked out by more than one user at a time, those files that are marked to be checked out by only one user, can only be opened by one

user at a time. the Perforce administrator can use a special table called the typemap table to automatically specify certain file types as exclusive-open.

3.5.2.1 Resolving Conflicts

If no locks were specified to a set of CIs, and multiple developers check out those CIs and change them, then at time of submits, conflicts are likely to be found. By nature, perforce will ask for conflict resolution by the later submitter. Because changelists are atomic transactions, until someone resolves the conflict, none of the changes to any of the files in the changelist can appear in the depot.

The resolve process enables the developer to decide if one file should overwrite the other file, or should a file be thrown away in favor of the other file's changes, or should the two conflicting files be merged into one file. At the developer's request, Perforce can perform a three-way merge between the two conflicting text files and the file from which the two conflicting files were derived.

The p4 diff and p4 diff2 commands produce output similar to that of the standard diff program included in UNIX and Linux systems. Other Perforce client programs (including P4V) include P4Merge, which provides a graphical view of file differences similar to that of the standard diff program included in UNIX and Linux systems. Other Perforce client programs (including P4V) include P4Merge, which provides a graphical view of file differences. [22]

3.5.3 Branches Creation

Branching in perforce, as in any SCM system, is used mainly to either test a newly-introduced set of CIs, or to develop a codeline with special requirements different from the original codeline requirements, or to release a new version of a product. The branching in perforce relate to physical storage, in a way that whenever a branch is created, a separate folder for that branch is created under the root folder and files are copied there. Then that folder evolves separately from its original codeline and can be merged back to the main codeline.

Perforce repository plays an important role in branching, since it keeps track of branching relationships. Hence the development team can view the history of changes that happened to each branch and its relationships with other branches changes, especially that copies for each branch are not really made, only referencing of shared files are kept and one copy of each in common file is stored.

Perforce uses branch on release convention and technology called Inter-File Branching technology to support this work. The support for branching in Perforce works such as to branch when an expected release has to do some special bug fixes from the main development code stream. As such compared to other source code management systems the main code-stream is

always considered the development stream whereas branches are made for releases and bug fixes. [22]

Branching in perforce also supports renamed and moved files. When it comes to altering the structure of the repository, it becomes difficult to track the origin of a set of CIs. The 'move' command branches originals to new files and deletes the originals. The system keeps track of file origins, however, and refers to them when displaying the history of renamed files. [52]

The administrator can create a manual codeline, in which a manual copy operation is carried out on a new folder. But the advantage of integration over copying the files is that files are integrate and tracked by perforce between branches, hence the team members can propagate changes to multiple related codelines automatically.

To integrate files from a source codeline to a target codeline, the target must be in the developer's client workspace view, and the source doesn't have to be in his client workspace view (although he must have permission to read the source files), then the user opens files for branch in a new changelist by integrating them, and then he creates files in the new codeline by submitting the changelist. When the user submits the changelist with the target files, the target files in the new codeline are at revision #1, and integration records enable him to examine the history of files in the new codeline, including the fact that they were created by means of integration from the source files. [22]

The developer can use integration to propagate changes between related codelines in the same way he creates codelines. When a user creates a codeline, the target files are by definition empty; there is no possibility that changes can conflict. When changes between existing codelines are propagated, conflicts can arise because conflicting changes may have been made in both the source and the target codelines. The process is illustrated in figure 3.6 [22]

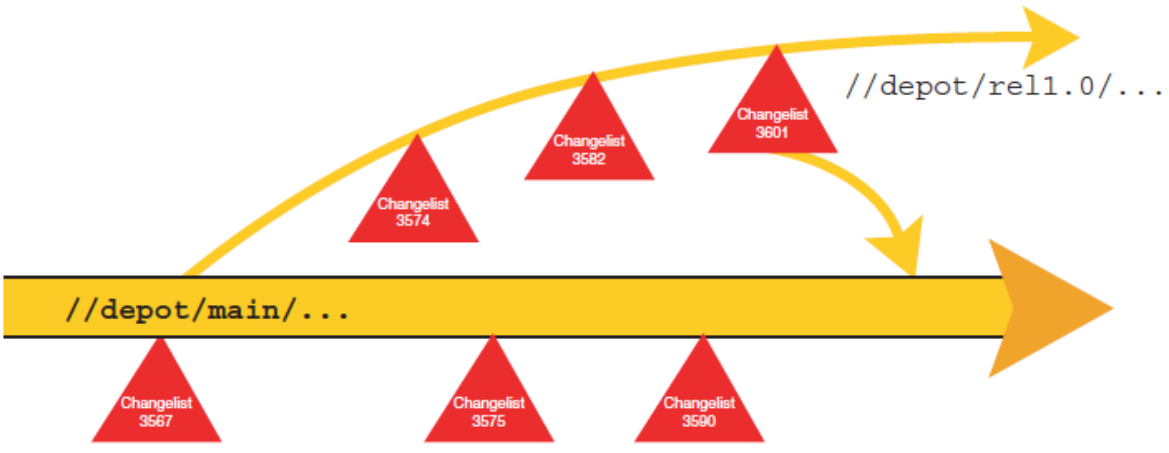


Fig 3.6: Propagating changes between codelines [22]

3.5.4 Work and Defect Tracking

Tasks in Perforce can be defined and assigned to certain resource for defect tracking purposes, Perforce includes a basic defect tracking system called jobs. A Perforce job is a description of work to be done, such as a bug fix or a change request. Perforce's job tracking mechanism enables the project manager to link one or more jobs to the changelists that implement the work specified in the jobs. Associating jobs with changelists helps teams know if and when work was completed, who performed the work, and what file revisions were affected by the work. Jobs linked to changelists are marked as closed when the changelist is submitted.

The types of information tracked by the jobs system can be customized; Perforce administrators can add, change, or delete fields used by Perforce jobs. Perforce offers two independent platforms to integrate with third-party defect tracking and workflow systems. Both platforms allow information to be shared between Perforce's job system and external defect tracking systems. [22]

P4DTG, the Perforce Defect Tracking Gateway, is a closed-source integrated platform that includes both a graphical configuration editor and a replication engine.

P4DTI, Perforce Defect Tracking and Integration, is an open-source product available under a FreeBSD-like license.

3.5.5 Tagging Files with Labels

Labels in perforce are used to collectively relate a group of files logically, labels in perforce basically depends on file versions and using it to connect file revisions to construct one logical unit. A label could represent a release for a certain customer, or a revised set of CIs waiting for some action to be completed. Labels differ from changelists in the collection of files that they contain; in a changelist, only CIs that were changes at the time of the changelist creation are included there. But labels can contain CIs that are from different changelists.

Another difference between changelists and labels is that changelists are referred to by Perforce-assigned numbers, while labels take names chosen by users. For example, you might want to tag the file revisions that compose a particular release with the label rel2.1. At a later time, you can update the revisions tagged with rel2.1 to reflect fixes performed in subsequent changelists, and retrieve all the tagged revisions into a client workspace by syncing the workspace to the label. [22]

3.5.6 Limitations of Perforce

Perforce is a light weight SCM tool, and is famously known for its performance compared to other SCM tools. Still it has some limitations that the company developing the product is trying to tackle during the ongoing upgrading of the Perforce SCM product, listed below are some of those limitations:

- Offline mode: the perforce SCM tool tend to rely heavily on the network that it is running on, and thus making the offline mode a bit scary to use, especially that through some reports recorded in the software vendor's website, users of the perforce system tend to have problems while working in offline mode and they are reporting unexpected results and behavior from the system while working offline
- Command line interface: perforce offers a command line interface beside the GUI based interface. But while using the command line interface, the user experience complications in dealing with it, especially with the commands themselves which are hard to memorize and get familiar with.
- Repository structure: the perforce repository structure is a bit trivial compared to more sophisticated SCM tools (such as IBM ClearCase). And although it is easy to backup and restore because of its simple design, it is rather hard to replicate in an environment that requires replicas of the database to be available.

3.6 ClearCase

Rational ClearCase is the most sophisticated software configuration management tool in this chapter. The scale of projects that this tool operates on is from medium to huge projects. It handles the most common configuration management procedures in its own defined methodology (to be discussed later). Also it runs on multiple platforms but requires large efforts during installation and operation.

ClearCase was developed by Atria Software and first released in 1992 on UNIX and later on Windows. Some of the Atria developers had worked on an earlier system: DSEE (Domain Software Engineering Environment) from Apollo Computer. After Hewlett-Packard bought Apollo Computer in 1989, they left to form Atria. Atria later merged with Pure Software to form PureAtria. That firm merged with Rational Software, which was purchased by IBM in 2003. IBM continues to develop and market ClearCase. [50]

As a popular commercial SCM tools ClearCase solves a broad range of SCM-related problems and provides both general-purpose and specific solutions. The specific solution, ClearCase UML model is based on two key concepts: activity-based SCM and component management. The activity-based SCM means: Instead of creating and manipulating a set of file versions, the consistent change under the activity-based SCM is treated as a single named entity. A set of versions is called a change set, an activity is a named object that identifies the specific task,

defect, feature, or requirement; and the component management is a method of breaking a software system into smaller manageable pieces, which is a mean of collecting a set of files and directories into a large entity that can be versioned and managed as a whole. [36]

ClearCase offers two configuration management interfaces: UCM and Base Clearcase, by which the user can create a configuration management solution suited to the organization's own development environment. [24]

The IBM Rational Team Unifying Platform (Figure 3.7) is a scalable SCM solution that allows organizations to start small with a simple SCM solution that enables them to grow without incurring the cost, effort, and disruption entailed in migrating to an enterprise SCM solution.

As illustrated in the figure below, IBM ClearCase offers multiple packaged solutions to meet the organizations expectations, requirements and budget. ClearCase LT is the most basic available package offered by IBM. It is mostly suited for individual workgroups. As the organization business expands and grows especially in network structure and development team areas, the organization can move to ClearCase which has the advantages of presenting multiple distributed servers into the development environment. And for huge projects with replications and multiple sites, ClearCase MultiSite is mostly suited. It has the ability of repository replicas over geographically distanced servers in huge projects development environments.

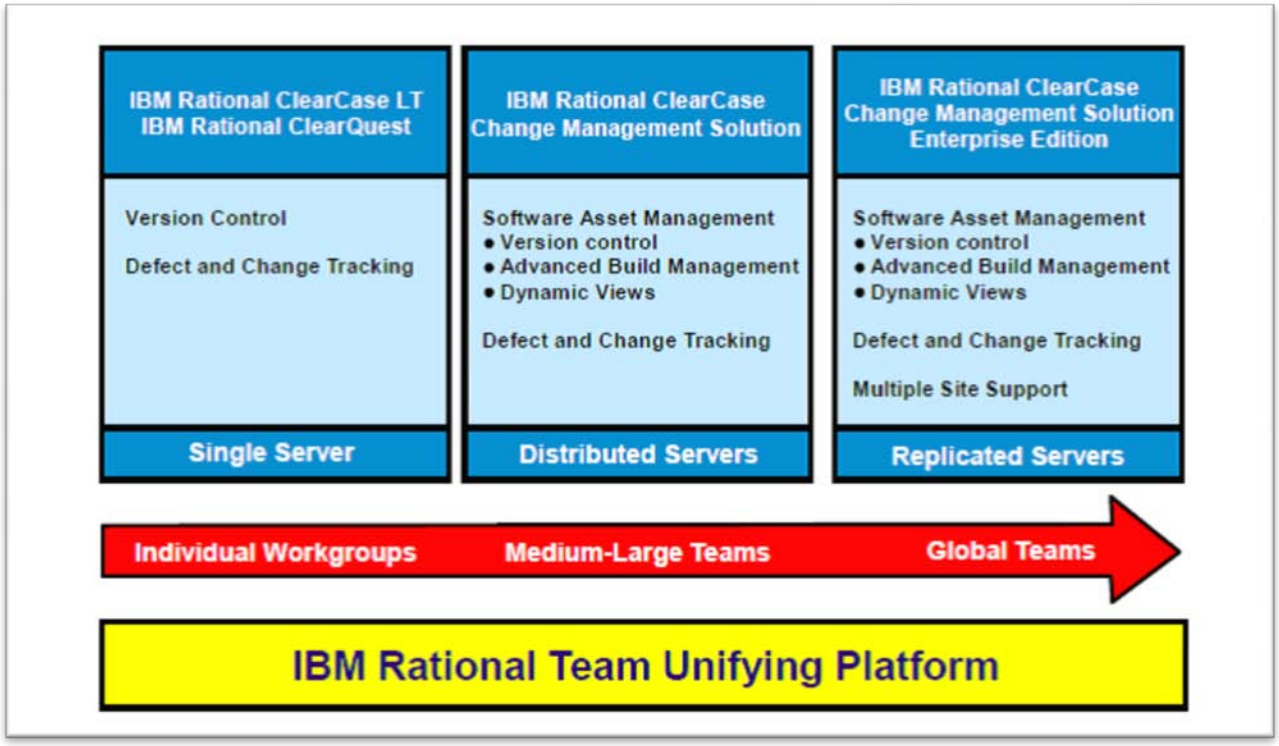


Figure 3.7: Rational available packages [34]

3.6.1 IBM Rational ClearCase Available Packages

IBM Rational Software delivers a scalable SCM solution that begins with ClearCase LT, but allows for a transition to enterprise-scale ClearCase and ClearCase MultiSite. It also integrates with IBM Rational Software's full suite of development productivity tools, including IBM Rational ClearQuest-a change and defect tracking tool-and Unified Change Management (UCM), IBM Rational's activity-based process for change management. [34]

Listed under is a brief description of each available package as provided by IBM documentation fo ClearCase [32]:

1- ClearCase LT

ClearCase LT is the basic version of ClearCase. It supports basic SCM functions and processes. ClearCase LT must be run on a single server, ClearCase LT versions CIs in the software development lifecycle. It tracks changes and maintains histories and enforces the organization's development process through scripted rules. Using ClearCase LT, developers can roll back to previous builds, baselines, or configurations.

ClearCase LT supports parallel development through automatic branching, which enables multiple developers to design, code, and test software from a common code base. It also includes ClearCase's diff/merge paradigm to automatically merge parallel branches.

Also, ClearCase LT supports project workgroups with process for controlling workflow through IBM Rational's Unified Change Management.

2- ClearCase

ClearCase simplifies the process of change in large development environments as it controls the evolution of requirements, models, source code, documentation, and test scripts. It handles version control, parallel development, workspace management, process configurability, and builds management. It also provides build auditing and a Web interface for universal data access.

3- ClearCase MultiSite

ClearCase MultiSite is very similar to ClearCase package in the functionalities available in both of them, but the only advantage that it has over ClearCase is that it enables parallel development across geographically distributed teams. It provides automated replication of project databases and access to all software elements and artifacts. It facilitates distributed development across both networked and non-networked sites with update mechanisms to support both network and tape transfers.

ClearCase MultiSite works in both on-line and batch modes, hence it has the ability to automatically resend information during network failures and recovering repositories in the event of system failure.

In Table 3.2 shown below, are the features available in each package of the ClearCase suite compared to each other.

Capability	ClearCase LT	ClearCase	ClearCase MultiSite
Low initial cost	✓		
Little training required	✓		
Easy to deploy	✓		
Support parallel development	✓	✓	✓
Support Unified Change Management	✓	✓	✓
Enforce policies and procedures through rules	✓	✓	✓
Rich functionality	✓	✓	✓
Customizable	✓	✓	✓
Flexible	✓	✓	✓
Protect code integrity	✓	✓	✓
Manageable	✓	✓	✓
Rich feature set	✓	✓	✓
Enforces development process	✓	✓	✓
Enterprise data storage support		✓	✓
Accelerated build management		✓	✓
Highly scalable		✓	✓
Support geographically dispersed teams			✓

Table 3.2: Features set for each ClearCase package [34]

3.6.2 Unified Change Management

Unified Change Management (UCM) is a change management technique developed by IBM and embedded in all ClearCase available packages, and it is ClearCase's main contribution to SCM. The lifecycle is activity-based in which it connects each change to an activity occurring in the project lifecycle, which gives the implementer of ClearCase a managerial perspective of change management.

UCM is available to ClearCase (all versions) and ClearQuest, and supports the development by raising the level of abstraction to manage changes in terms of activities, rather than manually tracking individual files. This enables every member of the development team to easily identify activities included in each build and baseline. With UCM, an activity is automatically associated with its change set, which encapsulates all project artifact versions used to implement the activity. This ensures that all code and content changes are delivered and promoted accurately. [34]

According to Wikipedia.org the Unified Change Management (UCM) is the object-oriented realization of ClearCase, a set of software tools typically supporting the process area software configuration management. UCM is activity based. The activity object is the basis for sharing information between ClearCase and ClearQuest. In ClearQuest, the activity presents itself as a task on the developers "to do list" query. In ClearCase, the activity connects the task with the actual files that need to be changed in order to accomplish the task (the change set). By looking at the change sets of those activities, it is possible to know every file that was touched in order to arrive at a new baseline. [51]

UCM enables the concept of parallel development with its own defined policies and practices and can be implemented in ClearCase. Also it assists development teams in creating and maintaining developer work areas, associating activities or components with specific software change sets, integrating project changes, creating and managing component baselines and using metrics to gain visibility to overall project status.

3.6.3 ClearCase UCM Main Components

With ClearCase UCM, development teams gain more abstract view of changes through connecting change to activities in the project's master plan rather than connecting changes to individual files in the project's repository. So basically each change set is related to a predefined task on the project's main master plan, UCM has basic elements that need to be reviewed, below are the main elements and their description:

- **Project**

A UCM project is a logical unit that is mapped to the development structure of an application or system. A project contains the configuration information (for example, components, activities, policies) needed to manage and track the work on a specific product of a development effort, such as an auction Web site or an order fulfillment process for an e-business. A basic UCM project in ClearCase consists of one shared work area and many private work areas (one for each developer). [34]

- **Component**

A component is a group of file and directory elements (source code and other relevant files, such as a customer GUI) that are versioned together. Components are considered as blackboxes, they contain interrelating inner subcomponents, and the component entity itself assembles them. The team develops, integrates, and releases a component as a unit. Components constitute parts of a project, and projects often share components. Components provide separation of concern and organize elements into entities. [34]

- **Activity**

An activity is an object that records the set of files (change set) that a developer creates or modifies to complete and deliver a development task, such as a bug fix. Examples of other activities include an update to a help file or the addition of a menu item to a GUI component. The activity title usually indicates the cause of the change. [34]

- **Branching**

The branch structures allow the development team to work in parallel on the same element. Some groups use this branching feature to work on more than one release of software at a time. UCM uses branches to provide private work areas (development streams and views) and the public work area (the integration stream and view). [34]

- **Work areas and streams**

A work area is a development area associated with a change. In Base ClearCase, a view is a directory tree that shows a single version of each file in the project, and the view is the developer's work area. In UCM however, a work area consists of a view and a stream. [34]

A UCM stream is the thing that defines for development a working configuration. It also allows for the working configuration's building, testing and deployment. It is defined by baselines or specific versions of set components or done activities within the environment. A stream is a ClearCase object that maintains a list of activities and baselines and determines which versions of elements appear in each developer's view. In UCM, streams are layered over branches, so that you do not have to manipulate the branches directly. [32]

A project contains one main project integration stream that records the project's baselines and enables access to the shared elements at the UCM project level. The integration stream and a corresponding integration view represent the project's primary shared work area. [32]

In UCM, projects development team has private work areas assigned for each developer. It contains the files that the developer needs to see, update and integrate with other developers' work. The project's main development stream is called integration stream, and it contains the assembly of the whole project's files in one codeline.

- **Baselines**

A baseline is a collection of some or all of the activities delivered to date. A baseline provides a common starting point for all developers on the project. The developer can rebase his development stream so that it contains the changes in the most recent recommended baseline. [34]

Each baseline contains multiple CIs from the project main integration stream, for each CI only one version of that CI is included in the baseline. The baseline marks an important event during the project development course; the project manager defines baselines to mark the occurrence of those important events with versions of CIs to reflect the type of the event that happened.

3.6.4 UCM Lifecycle

The UCM lifecycle begins with developers joining the project as defined resources, and consists of two main phases to go through, the first one is the development lifecycle and the second is the deployment lifecycle as shown in figure 3.8.

UCM is a layer built on Rational ClearCase to provide additional software configuration management features. These changes include integration with ClearQuest to enforce defect and change tracking with code development through the use of activities. This is part of the Rational Unified Process that describes the lifecycle of change management for IBM Rational's software development process. It also gives integrators ownership of projects and streams to allow policy and feature management by project leaders and release engineers. UCM is used and configured via either Command Line interface (CLI) or through Graphical User Interface (GUI). [51]

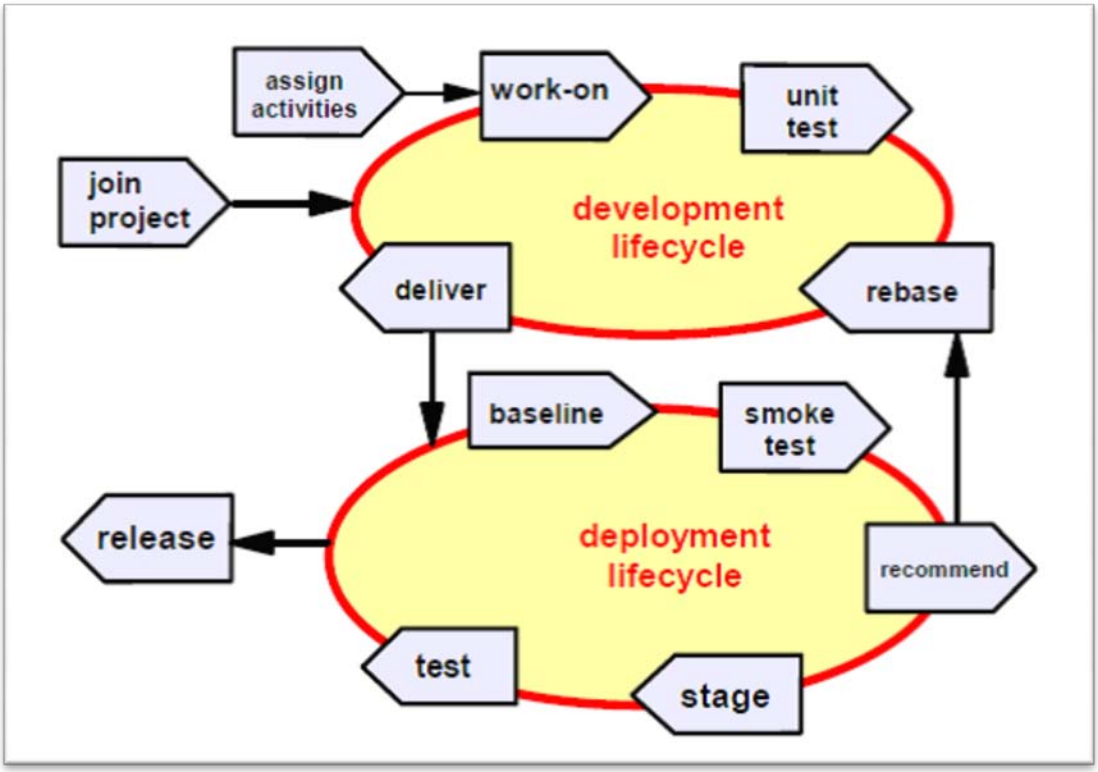


Figure 3.8: Typical UCM lifecycle [24]

Since the figure illustrates the workflow of the change lifecycle as UCM introduces it; table 3.3 shown below describes in detail each activity that was included in the figure above so a full understanding of the lifecycle can be constructed in the reader's minds.

Operation	Role	Description
Join project	Developer	A GUI wizard helps set up a working environment for participating in the UCM project.
Assign activities	Project manager	Assign work items (activities). Without ClearQuest the project manager has to find other ways to assign work to developers.
Work on	Developer	Make changes within the private development environment; every change is mapped to an activity.
Unit test	Developer	Verify changes before integrating with the rest of the project
Rebase	Developer	Synchronize working environment with the latest-best integrated configuration (recommended baseline) through a rebase operation.
Deliver	Developer	Commit changes to the shared project stream (integration stream) through the delivery of specific activities. By performing a rebase before delivery, the developer reduces surprises at delivery time, which improves the stability of the integration environment.

Baseline	Integrator	Select delivered activities and create the next integrated configuration (baseline).
Smoke test	Integration engineer	Verify that the baseline meets quality standards required to become the project's recommended baseline.
Recommend baseline	Integration engineer	Define the baseline as the latest-best integration configuration (recommended baseline). A baseline that does not pass the project's smoke tests is rejected.
Stage	Release engineer	Deploy software to test environments.
Test	Test engineer	Qualify configuration and submit change requests on failures.
Release	Release engineer	Deploy software to production environment

Table 3.3: description of UCM lifecycle phases [24]

3.6.6 Parallel Development

In some version control systems only one user at a time can reserve the access to a certain CI. In other systems, many users can fetch the CI and compete between them to create the next version. ClearCase supports both models by allowing two kinds of checkouts, reserved and unreserved :

- Reserved Checkout: Only one view at a time can have a reserved checkout of a particular branch. A view with a reserved checkout has the exclusive, guaranteed right to extend the branch with a new version. After the developer performs the checkin, he no longer has any exclusive rights on that branch. Another user can now perform a reserved checkout to claim the right to create the next version on the branch. [47]
- Unreserved Checkout: Many views can have unreserved checkouts of the same branch. Each view gets its own private copy of the most recent version on the branch; each copy can be edited independently of all the others. An unreserved checkout does not guarantee the right to create a successor version. If several views have unreserved checkouts of the same branch in different views, the first user to perform a checkin “wins” — other users must merge the checked-in changes into their own work before they can perform a checkin. [47]

By default, the checkout command performs a reserved checkout; the reserve and unreserve commands change the state of a checkout. Figure 3.9 illustrates checked-out versions created by reserved and unreserved checkouts, along with the effect of subsequent checkins. [47]

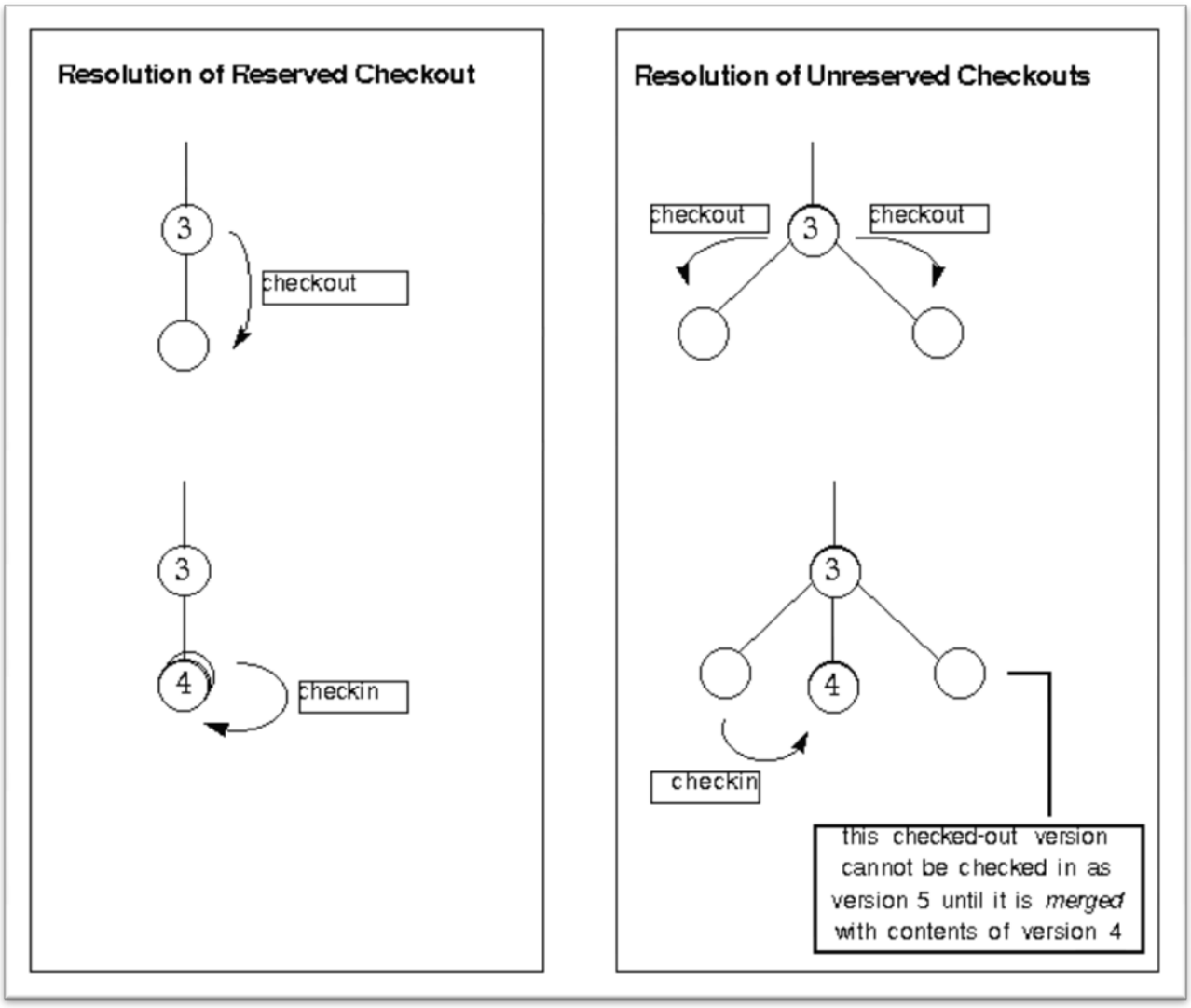


Fig 3.9: Resolution of Reserved and Unreserved Checkouts [47]

In concurrent development, we need a checkout model that allows for more than one person to check out a specific element at the same time. Such models can be either optimistic or non-optimistic checkout models.

3.6.6.1 Non-Optimistic

In a non-optimistic model, the order of checkout determines the order of checkin. Whoever checked out the specific element first, gets a reservation for first checkin. In Figure 3.10, developer Adam was the first to check out the libA.c element version 1.3, and thus was granted a reservation for first checkin. This means that even if Joe would try a checkin before Adam, he would not be able to do so, and he would have to wait until Adam had completed his checkin. [34]

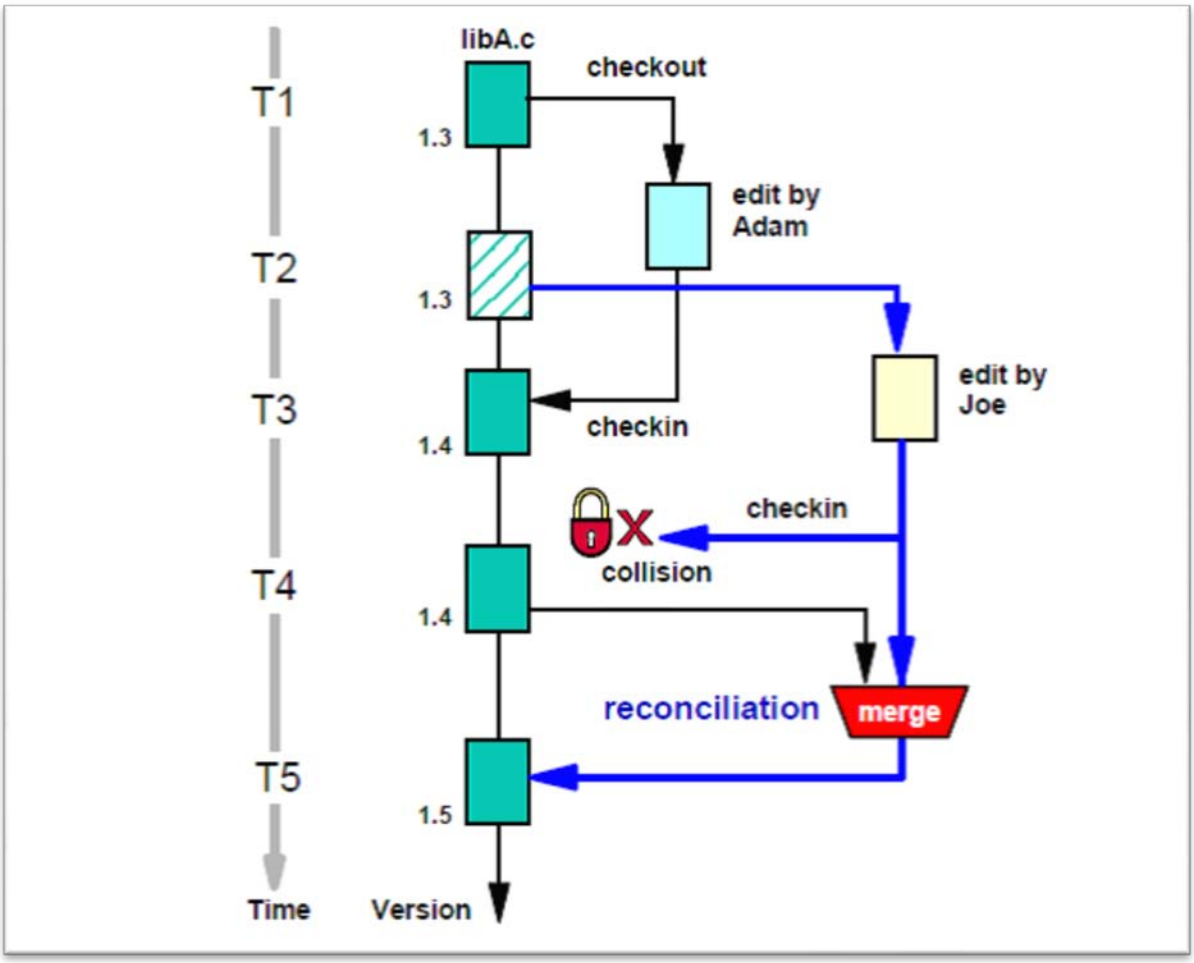


Fig 3.10: Parallel development checkout model [24]

When Adam's checkin is complete (version 1.4), Joe can go ahead with his checkin attempt. But because the same checked out version of the element has already been checked in by Adam, the tool will detect a collision situation and will force Joe to perform a settlement (of his edited version 1.3 and Adam's version 1.4). If the checked out element is a textual file, the reconciliation can probably be performed with the help of a merge tool. But for binary files, a decision has probably to be made as to which file will finally be checked in. [34]

3.6.6.2 Optimistic

In an optimistic model, a first-come-first-served policy is used, and the order of checkout is irrelevant. So even if Adam did the first checkout, as in our previous example, Joe could still perform a checkin before Adam. But as for our previous example, when the second checkin attempt is made, a collision is detected, and a reconciliation through a merge or a deliberate choice has to be made. ClearCase provides non-optimistic checkout by default, but can be configured to be optimistic per file or by default. [34]

3.6.7 Versioned Object Base

Versioned Object Base (VOB): A repository that stores versions of file elements, directory elements, derived objects, and metadata associated with these objects. With MultiSite, a VOB can have multiple replicas, at different sites. [51]

A VOB stores file system objects: directories, files, links. (It also stores non-file-system information, meta-data). ClearCase VOB implements a technique called storage pools; A VOB storage pool is a subdirectory that stores users' file system data. Some storage pools provide a repository for historical and currently-used versions of source files. Other storage pools provide a repository for object modules, executables, and other derived objects created during the development process. [32]

ClearCase can store individual versions of files in several ways, using such techniques as data compression and line-by-line deltas. The data storage/retrieval facility is extensible which means that users can supply their own type managers, implementing customized methods for storing and retrieving development data. [sgi01]

Another feature in the ClearCase VOB is that the whole repository can be segmented into portions to separate different projects from each other; each portion in this case is called a Project VOB (PVOB) which is a special type of VOB that is used when UCM is implemented as the software configuration management process. Every UCM project belongs to a PVOB and multiple UCM projects can share a PVOB. A VOB family (which exists in ClearCase MultiSite) terminology is used in geographically apart environments and when replicas exist, VOB family is the set of all replicas of a particular VOB.

VOBs support the notion of triggers, which are attached to elements and specify one or more standard programs or built-in actions to be executed whenever a certain ClearCase operation is performed on the element. [34]

3.6.8 Limitations of ClearCase

Performance: ClearCase has two modes of operation, snapshot and dynamic views. But the problem is that dynamic and snapshot views are slower than local filesystems, even with good network infrastructure. Because MVFS requires server access every time a file is accessed, the performance of the file system depends on server capacity. The ClearCase protocol relies on

remote calls for most actions which slows down even snapshot views, compared to other SCM tools (e.g Subversion). [51]

Sensitivity to network problems: Since ClearCase has two modes of operation, snapshot and dynamic views. Dynamic views are essentially online view so any problems with the server or network render the dynamic views unusable. It is not possible to work on dynamic views offline. However, the snapshot view can work offline efficiently if the view data for dynamic views is stored on a client host (as may be done for performance reasons or to reduce the need for servers). [51]

Cost: ClearCase has the highest prices between all other free, open-source SCMs and commercials SCMs. The costs of ClearCase also include licensing, administration and hardware costs. [51]

3.7 Visual SourceSafe

SourceSafe(VSS) is a software configuration management tool, which is file-system-based source control tool from Microsoft used by many Windows developers.This tool was originally developed by One Tree Software Company and later taken over by Microsoft. [VSSGAO]

SourceSafe provides for true project level configuration control. In 1995, SourceSafe was taken over by Microsoft and re-named. According to their sales office, Microsoft added conversion utilities from Delta and PVCS. The 4.0 release includes support for long filenames and UNC paths, a tab dialog for setting options, localization into 5 languages, a Windows95 look and feel, and tight integration into Visual Basic, Visual C, Visual Test, and Fortran PowerStation. [VSSFAQ]

Starting with VSS 2005, Microsoft has added a client–server mode. In this mode, clients do not need write access to an SMB share where they can potentially damage the SS database. Instead, files must be accessed through the VSS client tools - the VSS windows client, the VSS command-line tool, or some application that integrates with or emulates these client tools. [VSSWIKI]

As per Microsoft MSDN description for VSS, the following is the basic description for VSS through Microsoft's perspective:

"Microsoft Visual SourceSafe is a file-level version control system that permits many types of organizations to work on several project versions at the same time. This capability is particularly beneficial in a software development environment, where it is used in maintaining parallel code

versions. However, the product can also be used to maintain files for any other type of team. [msdntfs]

SourceSafe keeps all versions of the project filesin the whole workspace that developers had checked in (if “Store only latest version” option is not selected which will exclude the selected file from versioning control). And under normal circumstances, you see only the latest version, which makes your file management much easier. Whenever you need previous versions, you can use the “Show History” feature to access all the previous versions.

VSS has a model for setting up multiple versions of a project. The key commands are the share, branch, merge, links, and paths commands. Rather than using numbers to branch, such as version 2.3.6.1 in SCCS, a logical release or customer name can be used to implement the same construct. SourceSafe also runs on many platforms so it can be used for a client/server project where coding is being done on a Windows PC using Visual Basic, and on a UNIX workstation using C.

3.7.1 Files Diff

VSS can visually diff non-binary files, such as your C#, ASP.NET or Java source code. Word and Excel files are binary files and cannot be diffed in SourceSafe. You can view the differences between local file and any version in the VSS database, two previous versions in VSS, any two local files or any two files in the VSS database, You can see which lines are newly added, which lines are deleted and which lines are changed. [VSSGAO]

3.7.2 Project/Folder Difference

Project difference feature allows you to see the differences between your local folder and VSS project, two local folders or two VSS projects, the diff result shows you which files are only in the local folder, which files are only in the VSS database and which files are different. The developer can do version control operations, like check out, get and view, directly in this interface. There is also a powerful feature called Reconcile All, which can synchronize your whole local folder and VSS database. [VSSGAO]

3.7.3 VSS Database

A VSS Database is a single instance of a VSS server – it is the big black box into which files get placed. All commands to VSS are directed towards a particular VSS Database, and wach database maintains a SRCSAFE.INI file which contains configuration information. [VSSDNS]

3.7.4 VSS Project

A VSS Database is organized as a tree structure, with each of the nodes of the tree being a VSS Project. Each database contains a single root project, which can branch (to a depth of 15 nodes) into sub-projects.

VSS Projects are misleadingly named; instead they should be thought of as directly analogous to filesystem directories, being unordered collections of up to 8000 files of any type. To illustrate this, note that where an application's source-code is organized into files that live in subdirectories off the main directory, these subdirectories have to be mapped onto subprojects of the application's main project directory. [VSSDNS]

3.7.5 Working Folder

Because the files stored in VSS are not directly exposed as files, development work on these files takes place on local copies, which are first checked out of VSS, changed, then checked in again. While a file is checked out of VSS, it can be locked to other developers, preventing file overwriting. VSS also retains historical information about a file's changes, so it is possible to extract and use old versions of a file, or roll back unsuccessful changes.

The folder in which a user manipulates his local copy of VSS project files is known as his 'working folder'. Each user will often have a distinct, local working folder for each VSS project, the details of which are held by the VSS client in its SS.INI file. [VSSDNS]

3.7.6 Building and Deployment

For non-web applications, their compilation and deployment is not under the control of VSS. The process that builds and deploys an application needn't integrate with VSS except to extract the latest instances of the files into a local directory where they can be manipulated as required.

For web applications, VSS does support a 'deploy' command. This copies a project's files onto a webserver, either directly or via FTP. However, for various reasons one might choose to use other methods of deployment with web applications. [VSSSSS]

3.7.7 Multiple Checkouts

By default, if you check out a file then no other user can check it out until you've checked it back in: the file becomes locked to changes by anyone other than you. But the VSS administrator may enable 'multiple checkouts', which (fairly obviously) allows multiple users simultaneously to check out files.

So, what happens when UserA and UserB each checks out a file to his local working folder, each updates it independently, and then each tries in turn to check it back in? What needs to be avoided here, obviously, is UserA's changes being overwritten by UserB's check-in. What VSS tries to do is to merge each user's changes into the master copy. This is easiest if users have made

small changes to different parts of the file. However, if users have each changed the same part of the file then merging is not possible, so VSS rejects UserB's check-in attempt, with appropriate information about the problem, and requires him to resolve the conflict. [VSSSSS]

3.7.8 Limitations of VSS [VSSLIMIT]

1- SourceSafe badly handles slow networks and the public internet, SourceSafe is unusable over slow network connections. It is effectively unusable over the public internet. In addition, because SourceSafe works over network shares, if you place a SourceSafe server on the internet, you're exposing any weaknesses in your servers file sharing implementation to the entire world. Of course, if you are willing to invest more money in your ineffective revision control system, you can buy a third party product to solve this problem.

2- Managing third party modules is difficult with SourceSafe, It is not uncommon for a developer to use third party modules in your project to quickly add required functionality.Unfortunately, SourceSafe makes tracking a third party module extremely difficult. Initially checking the first version in is not hard. Checking a new version in requires a good memory and attention to detail. To add a new version, you first recursively check the folder holding the module out. Now delete the directory on disk and replace it with the new version. Check new version in.

3- SourceSafe relies on dangerous file sharing, SourceSafe doesn't really run as a server, but as a set of files shared over SMB. As a result, you're relying on each individual client to not misbehave. A single misbehaving computer can destroy the database. A problem in the file sharing implementation on your operating system can damage the database. Users only need read-only access to the revision control system need write access to the server, increasing the risk.

4- SourceSafe degrades on large projects, Microsoft recommends that your database not exceed 5 GB. While this is a large database, it's not unreasonable for a large project, especially if you check in large binary files (like Microsoft Word documents).

5- Viewing and retrieving historical versions is extremely slow, It's not unusual to need to get a historical version of the source code. You might need an older version to investigate a bug report, or the current code is malfunctioning and you need to get a functioning version. SourceSafe supports this, but it's extremely slow for non-trivial projects. To get a historical version, you first need to generate a history for the entire project you are interested in.

3.8 Team Foundation Server

Team Foundation Server (commonly abbreviated to TFS) is a Microsoft product offering source control, data collection, reporting, and project tracking, and is intended for collaborative software development projects. It is available either as stand-alone software, or as the server side back end platform for Visual Studio Team System (VSTS). [wikitfs]

The primary purpose of Team Foundation is to enable collaboration on a team to make it easier to build a product, or complete a project. There are many types of projects. Software projects involve building and releasing a software product that is typically a new product, an upgrade to an existing product, or a minor update release. [msdntfs]

Team System can be viewed as three logical tiers: the client tier, which includes the Team Editions of Visual Studio 2005; the application tier, in other words, Team Foundation Server; and the data tier (SQL Server 2005), which provides the data management and storage support behind the scenes. This very basic architecture can be seen in Figure 3.11. [PTFS]

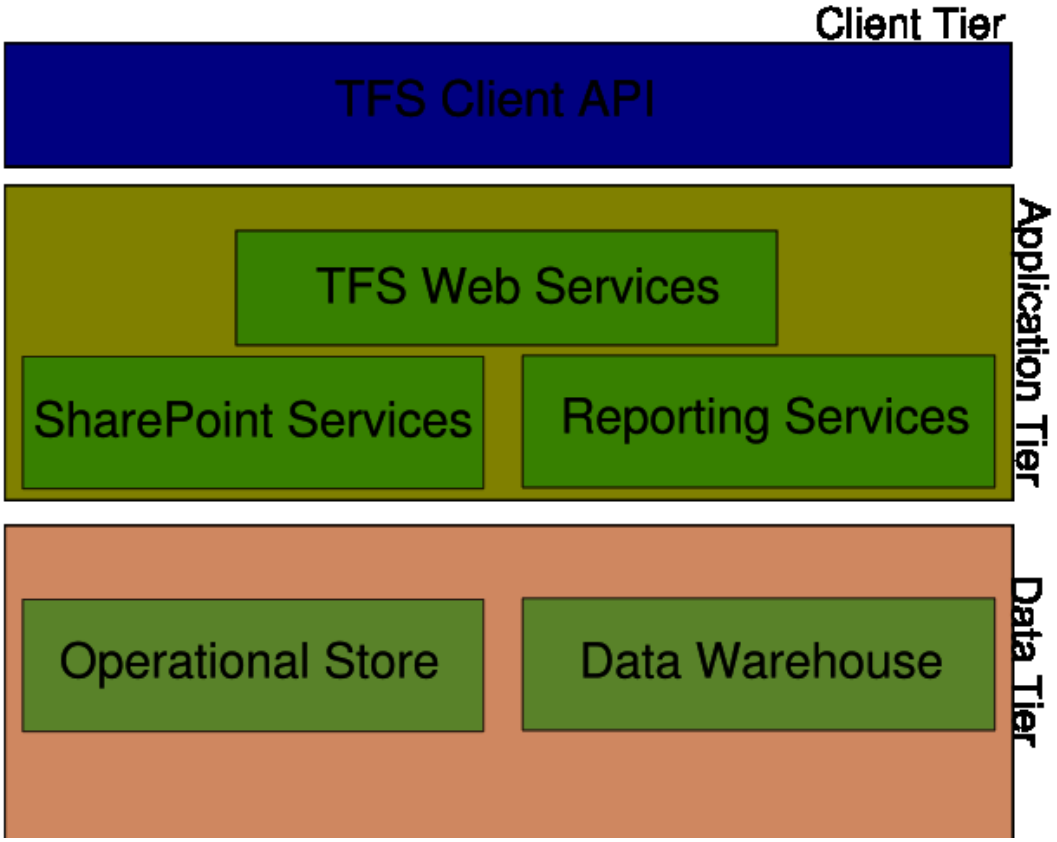


Figure 3.11: TFS three-tier architecture [wikitfs]

3.8.1 Source Control

During the project course, team foundation server manages and stores mainly the source code of the project and maintains version history of those files. But typically the team would in most cases include other project-related documents in the source control repository to be managed by the team foundation server.

The basic functionality in any version control system includes checking file in and out. To support concurrency, TFS allows multiple checkouts of the same file, but this can be disabled should the need arise. Items can also be exclusively locked so that nobody else can check in or out a file while it is locked. If concurrent checkouts are disabled in team project settings, then a lock is automatically placed on the file upon checkout. [codegurutfs]

Branching and merging can be considered advanced functions in TFS, but nonetheless, they are highly useful. The main idea of branching is to take a set of source code files and create a distinct version from those files. The branched code can live a life of its own separate from the original source files. For instance, if you are developing a generic application but need to make a specialized version for a certain customer, you could branch the customer customizations from the main source control tree (the "trunk"). Should the need arise later; you can again combine the customization code with the original source control line. This is called merging. [codegurutfs]

There are two primary types of roles that use Team Foundation source control. The first role is a contributor is a user of Team Foundation source control who is primarily concerned with adding, deleting, and modifying files in the source control server. This role is most frequently associated with software developers working on a software project; however, because not all files that are contained in the source control server are necessarily source code files, the contributor role is not limited to developers. For example, a contributor could be a technical writer whose product is documentation, or a graphic artist whose product is a binary image file. [msdntfs]

The second role is an administrator who is concerned more with the administration of the source control server and the use of the files that it contains for creating a reproducible software build. An administrator is responsible for maintaining the integrity of data stored in the software configuration management system. This task differs based on a particular system; however it usually involves managing access to the source control server and enforcing a backup policy for data that it contains. Because the software configuration management system is the storehouse for the intellectual property of the company, the administrator has ultimate responsibility for ensuring the availability and the integrity of the corporate intellectual property.[msdntfs]

3.8.2 Work Items

Work item is another major object that teams use to prioritize and track and work. Work comes in many forms: bugs, tasks, features, requirements, change requests, issues, and so forth. Most

teams today need to deal with several different tools to make sure they are getting a clear picture of everything that needs to be done. Team Foundation Server has the flexibility to manage all of these types of objects as different work item types, but tracks them in one place, allowing for cross work item type reporting and queries. All work items include things such as a description, assigned user, and states for tracking progress. However, with Team Foundation Server, work item types are completely customizable, which allows your team to define the fields, rules, state model, and form layout that make sense for your team. [msdntfs]

3.8.3 Workspace

TFS Workspace needed when you do checkin/checkout and any other operations that edit existing file. When you try to edit file for example, TFS checks your local version of the file with the TFS Source Control and updates it with the latest version before letting you edit it. On the other hand, when you try to checkin pending changes; TFS looks at you local workspace and checks against its own version. If both versions have conflict, TFS prompts you with the conflict resolution screen that you use to resolve any pending conflicts. Once everything is resolved. TFS takes your local copy of the file from your own TFS workspace, creates changeset and commits changes to its own version of the source file. [VSTSTFS]

A TFS Workspace contains several important bits of information. A workspace is identified by its name and its owner. The workspace name can be up to 64 characters in length, and the name must be unique for a given owner. The workspace also contains the hostname of the computer that the workspace is associated with and there is room for you to give your workspace a comment (a good idea when you have multiple workspaces so you can remind yourself what they are for). A workspace also contains the working folder mappings – the links between your computer's hard drive and the TFS version control repository. You can also picture the workspace as the way in which the TFS server keeps track of which files you have downloaded, what versions of those files you have and which ones you have pending changes and / or locks against in your local file system. [WWWTFS]

3.8.4 TFS Limitations

1. Storage limitations: Team Foundation Server depends on SQL database which is also limited by the amount of data that can be stored in it. The maximum recommended amount of data for each component of the Team Foundation Server has been identified by Microsoft on MSDN website. [msdntfs]
2. Performance: Team Foundation Server decreases performance and efficiency dramatically as the number of clients and projects increases. [tfsblog]

3.9 ChangeMan

The Serena ChangeMan Professional Suite, is a software configuration management (SCM) that automates, accelerates and manages development projects. The Professional Suite is comprehensive, combining products for version management, defect management, build management and project issue management in a single package. [serenapo]

Serena ChangeMan solutions tackle the complexity of developing, deploying and maintaining software applications. ChangeMan can manage parallel changes to all applications, regardless of development methodology, geographic location, or computing platform. As a core element of the Serena Application Framework for Enterprise (SAFE), ChangeMan solution enables users to automate, control and synchronize changes across the enterprise from a single point of control. [TechDirect]

3.9.1 Serena ChangeMan Builder Product Editions

1. **Serena ChangeMan Builder Standard Edition**: Designed and optimized for Windows operating systems. It comes standard with Serena ChangeMan Professional Suite for SCCM, and with Serena® ChangeMan® Dimensions™ for SCCM and IT Infrastructure Management.
2. **Serena ChangeMan Builder Enterprise Edition**: Provides automated build management to all development platforms and environments; adds support for Java, impact analysis, and remote builds. Available as an upgrade to the Standard Edition.
3. **Serena ChangeMan Builder for z/OS**: For use with Serena ChangeMan Dimensions only, extends application builds to support the IBM mainframe environments: OS/390 and z/OS.

3.9.2 ChangeMan Version Controller

ChangeMan Version Manager organizes, manages and protects software assets during distributed team development. Application quality is improved and productivity increases, as Version Manager automates common tasks and protects against lost changes, overwrites and content errors. More than simply storing file versions, Version Manager enables and automates complex team development tasks such as parallel development, visual differencing, branching and merging, identification of merge conflicts, promotion levels and team workflow. [serenapo]

For the project manager, ChangeMan Version Manager makes it easy to assign tasks across distributed teams and remote developers, regardless of their development environment. Common, unifying version control practices can be established across distributed teams,

resulting in higher team productivity, cost savings and less rework in ongoing projects. [serenapo]

For developers, Version Manager offloads development task management and revision control, so they can maintain focus on creativity in their own programming. Build managers find fewer coding errors halting builds when developers use Version Manager to boost code quality. In addition, It integrates with ChangeMan Builder to assure reliable and automated build processes that give the QA team more time to test, with fewer delays due to build errors. [serenapo]

QA engineers and software test engineers must verify changes and ensure that the changes are approved, support the requirements, and can proceed into production. With Version Manager and its web client, QA and test personnel can stay on top of projects or validate a specific module, even if traveling to a remote development site or while working from home. [serenapo]

3.9.3 Change Management

Serena TeamTrack is the automated way to capture, track and manage the status of defects, changes and issues raised in team development. Notification rules and the ability to organize and handle multiple issue types (including workflow with parent/child relationships) assure that TeamTrack is more than an email history of development assignments. TeamTrack enables the team to analyze issue dependencies, see progress, spot issue trends and more effectively assign team resources to defect resolution. [serenapo]

A change package consists of descriptive information, control parameters, component metadata, history information, and a set of libraries that belong exclusively to the change package and which contain the software components being changed as illustrated in figure 3.12. A change package is a secure development environment for project components, with access managed by ChangeMan ZMF using rules stored in your security system. They make mainframe development much simpler. [SCMTCM]

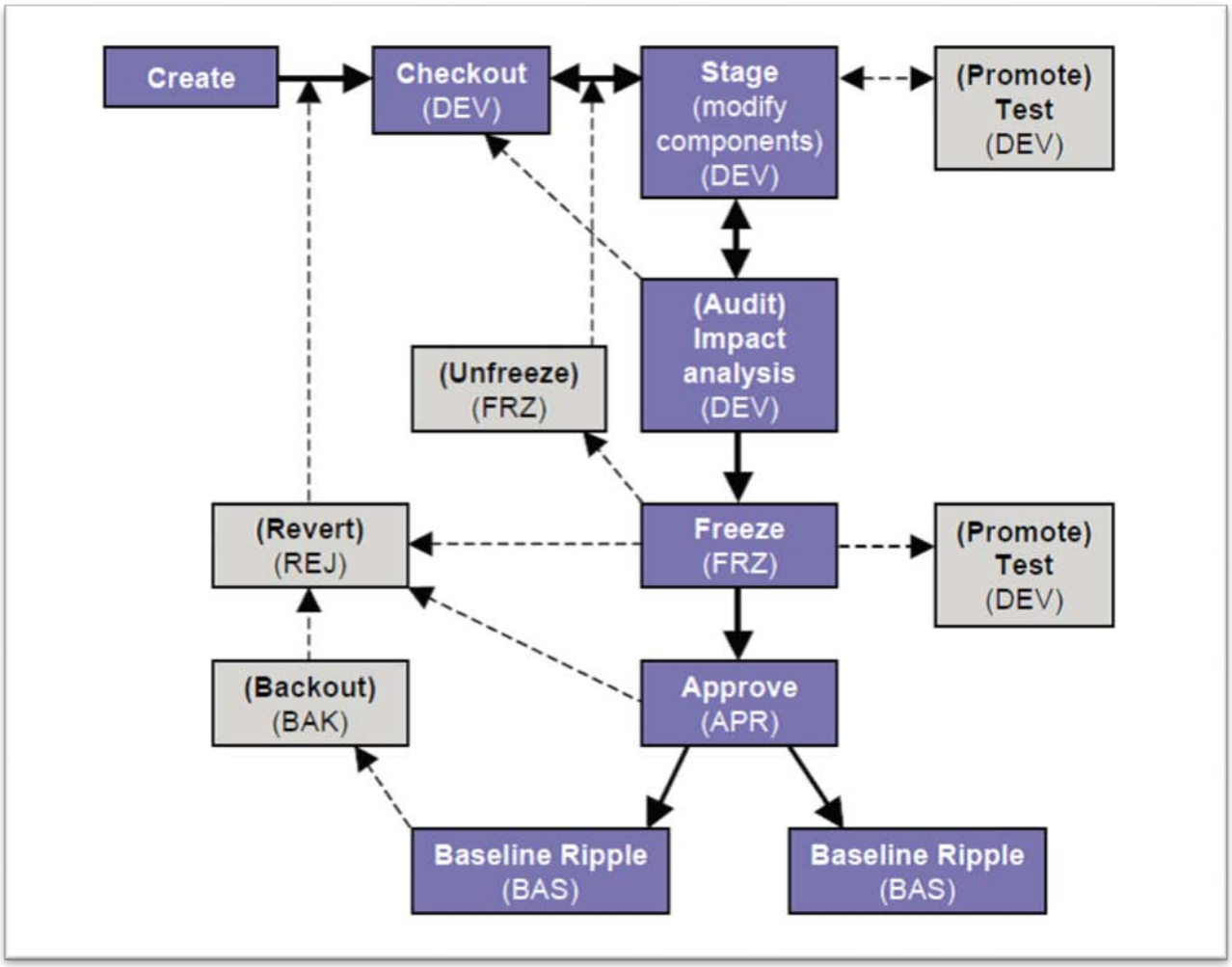


Figure 3.12: ChangeMan ZMF lifecycle [OVUM]

From the very beginning, ChangeMan ZMF has based the development process upon the Change Package. This intuitive approach to software development, invented by Serena, creates a unit of work within a software development or maintenance project that brings together all affected artifacts. It is managed as a unit throughout your defined lifecycle while software changes are in motion. The flow of packages through your system is designed around how you do software development. For example, if you need a special process for emergency fixes, ChangeMan ZMF will support that. Perhaps you need a different lifecycle for the online programs versus the batch programs; ChangeMan ZMF supports that, too. Maybe the development task is vast and you want to organize the changes into groups based on geography, functionality or technology. Yes, ChangeMan ZMF supports that, too. [SCMTCM]

3.10 In-Common SCM Tools Limitations

We have reviewed multiple SCM and VCS tools during the course of this chapter, and listed limitations of each tool as observed by the tool implementers. But most of the limitations discussed are non-functional limitations, which includes performance issues, tool becoming unresponsive in certain situations, network issues, etc…

So we had to thoroughly examine the methodologies that those tools work on, and discover limitations and shortages and sometimes inconvenient approaches to SCM practices. And we discovered some limitations and shortages in common between all of the reviewed tools. Those limitations are illustrated below:

3.10.1 CI Identification

In the field of CIs identification in SCM, tools give each item a unique identifier to ease the process of identifying and updating it. Also advanced tools such as ClearCase gives the user an option to add a clear and understandable name for the CI. But mostly those names assigned by the user are duplicated more than once in the SCM repository and the name does not differentiate between the revisions of the CI itself. For example, a user gives the name “Home Page” to a web page file, but the file keeps changing and the version number increases, which results in multiple instances of the same file having the same identifier. Most SCM tools do not offer a user defined name that would be kept unique across the whole SCM repository for that project. Also tools do not provide naming conventions for CIs to be defined to adapt to the organization’s development environment.

3. 10.2 CI Locking

SCM tools discussed earlier do not provide the option to lock a certain group of CIs within the project’s repository. Also this option can be done by altering the CIs access permissions for each individual CI, but this methodology is rather frustrating. The main purpose behind CI locking is being able to preserve certain set of CIs from accidently being altered, either for temporary or permanent period especially if the project contains blackboxes or third-party components.

3. 10.3 Baseline Locking

SCM tools discussed earlier fall short when attempting to lock a certain baseline. The main purpose behind baselines locking is that it helps maintaining a certain release for a certain customer in its original state. Also the baselines locked can limit the confusion that occurs in

huge projects while observing the project's evolution, since locked baselines can be removed from the big picture of an active project.

3. 10.4 Baseline Presentation

Baselines are the main milestones of the project, hence the project manager should be able to recognize the process on the project development by using baselines. But Most SCM tools consider baselines as a set of CIs with certain version numbers, alongside the information that was provided by the creator of the baseline. This methodology does not give much visibility to the managers of what were the differences between each baseline and other baselines.

3. 10.5 CI Versioning

SCM tools tend to change the version numbers of CIs whenever a change happens to them, this is the main purpose behind implementing such tools. But what if the change was a simple bug fix? Then an inconvenient situation would occur, since the version number would increase, the developer undergoing the change would go through the process of change approval and evaluation. Hence the project might end up having a series of changes to a certain CI with increasing version numbers, but with no real modifications added to that CI (only bugs being fixed).

3. 10.6 Conflict Resolution

When two or more updates conflict and no automated resolution could be applied, SCM tools discussed earlier opens the files conflicted in an editor and highlights the conflict areas according to a certain algorithm. This algorithm, depending on the tool itself, could be line-based or character-based or some other algorithm. Those tools tend to apply only one algorithm to every kind of CIs in conflict resolution. This practice could be a waste of time for the developer, since he might have binary or image files as the conflict subject, and the tool has an algorithm with line-based comparison to offer, which is highly frustrating in this situation.

3. 10.7 Change Management Lifecycle

Each SCM tool has its own defined change management lifecycle, and that lifecycle control the change process to the project's CIs. But the main problem here is that SCM tools do not offer the flexibility of controlling changes according to its effect. Changes could be either major or minor or simply a bug fix, the earlier discussed tools handle those three situations in an identical manner, which can be very disappointing and decrease the work efficiency of the development team because of the time wasted during unwanted controlling overhead.

Chapter 4

The Proposed Model for an SCM Theoretical Tool

In chapter 2 we introduced the concept of the software configuration management, and its basic components according to world-wide accepted standards and best practices. In Chapter 3, we reviewed multiple version management tools and software configuration management tools; during those reviews we tried to expose their weaknesses and strengths according to the software vendors and implementers point of views.

In this chapter we illustrate our proposed model based on the facts collected from the previous chapters regarding the efficiency of the available SCM tools and standards in the IT market nowadays. The model is designed to tackle the unsatisfying features and work flows to gain better usability and satisfaction of SCM tools.

4.1 The Proposed Model

In this section and the following sections, we illustrate the proposed model for the theoretical tool of SCM. The proposed model contains theoretical alternative methodologies to solve the shortages and weaknesses observed while discussing the concepts and tools of SCM.

The proposed model mainly discusses the identification of software artifacts during the course of the project, and adds attributes to CIs to easily identify them within the project database. Also in the field of baselines, it redefines how the tool should manage baseline representation to better usage of baselines in the project.

In the field of version management, the model uses a hybrid approach between the most used approaches (lock-modify-unlock and copy-modify-merge). Each approach is used whenever it is mostly suited depending on the change type and impact on the project.

And for the main part of SCM, which is change management, the model differentiates between each change type by creating a dedicated form for it, and by redesigning the lifecycle to adapt to the change type, impact and priority.

We begin our illustration of the model by comparing the main concepts of SCM framework against our proposed methodologies in each major SCM area.

4.2 Artifacts Identification

To do configuration management, the first step is to identify which software artifacts should be placed under configuration management system control. These artifacts should include both those used to manage and design a system (such as project plans and design models) and those that instantiate the system design itself (such as source files, libraries, and executables and the mechanisms used to build them). IEEE calls this configuration identification: "an element of configuration management, consisting of selecting the configuration items for a system and recording their functional and physical characteristics in technical documentation". [16]

In terms of an SCM tool, identification means being able to find and identify any project artifact quickly and easily. In an SCM-enabled environment, the main purpose of CIs identification is to know what version of what CI to include in a certain system configuration. Failing to accomplish the task would cause a chaos in the project and false system variants would appear since choosing the right components failed.

The SCM tool tracks the CIs and their definitions and description during the project course. And the organization should handle the naming of those CIs to be able to easily refer to them when necessary.

Identification is an essential function provided by SCM. Thus, each software object carries an object identifier (OID) that serves to identify it uniquely within a certain context. In our proposed model there is an extra attribute that was introduced to CI identification which is “external OID”.

The organization could develop its own naming convention and define it onto the SCM tool; by that we define a naming strategy for each kind of CI and the tool must have the flexibility to learn the conventions from the administrator of that organization. (e.g. document could have the following naming convention “doc + [tool generated number] + [document name from the owner] “.

The “tool generated number” is a serial number defined by the tool for each newly introduced object to the system. And the “document name” is a convenient name that describes the content provided by the object owner. The combination field will form the Object Identifier (OID) for the object.

External Object Identifier “EOID” can be assigned by the owner of the artifact, to easily identify each object in the SCM repository using a meaningful name (e.g. Home_Page). Alongside the EOID is a system-generated, unique OID which may be used internally by the system. Both attributes are unique across the project’s repository, but the only difference is that an EOID can be renamed and does indicate a meaningful name.

The EOID can be helpful in interchanging objects between users and other users and can be changed at any time. Also it simplifies searching for an object. Another feature that can be

optional in the proposed SCM tool is that after a baseline has been defined, the EOID can also be locked for any further changes except by the original owner of the CI. The extra attribute (EOID) defined by our proposed model would enhance the process of the artifacts identification and would make it easier to manage and create them for the system users.

4.3 Baselines

During developing a software, the organization recognizes some certain milestones that need to be recorded for future references. Baselines are used to create those references.

SCM tools reviewed earlier create baselines using CIs versions and notes from the creator of the baseline. This means that each baseline contains files references with their respective version management identifiers. But if managers need to evaluate the differences between one baseline and another, they need to get back to the changes made to the included CIs and compare them to each other, in order to gain information about the real modifications that happened between those baselines.

Also once a baseline has been created, no restrictions to the previous baselines are involved. The developers can still see what previous baselines were and can still edit earlier CIs included in previous baselines, although they might be obsolete and mostly no longer in use.

Hence the best practice here (which is introduced in our proposed model) is to enable baselines locking for project managers as an added privilege. If a certain level of development and stability has been reached, and a desire to prevent further changes has arisen; In this case, a basic rule of SCM systems has to be overridden which is the encouraging of change, and the system would be locked for changes under certain version number or by reaching a certain functionalities sets that were approved and tested by the system quality assurance committee.

If any further requirements for change in some particular modules have arisen after the baseline has been locked, the project manager has to initiate another branch of code to satisfy the new requirements (e.g. special components for certain customer that differs from the baselined software configuration that was approved for being released).

Regarding the baselines creation process itself, we have integrated version and change control, so that baselines are no longer defined by CI versions, but as the previous baseline plus (or minus if rolling back) a number of changes, for example:

Baseline Y = Baseline X + Change A + Change B + Change C.

Illustrated in Figure 4.1 below are the operations for baselines determination and calculation in our proposed model.

Figure 4.1: baselines in our proposed model

In our proposed model, the baseline is defined by the previous baseline plus the change requests that have been applied after the creation of the previous baseline; this approach was inspired by the file differencing algorithm (deltas) that was introduced in the previous chapter.

In case of backward mapping of a baseline, the system does reverse the transactions that happened to the later baseline, and rolls back the change requests that were applied to the later baseline to gain the previous baseline. During the rolling back of the change requests, none of the change requests nor the new versions would be removed, only the artifacts included in the roll back operations would be reversed to a previous state, but the latter state would still be available and can be forward tracked using the previously described operation.

The main advantage behind the baseline differencing is to gain more visibility in the project, since all baselines are defined by change requests and not by version numbers. The new definition of the baselines is far more appreciated especially by upper management, and since each change request has its own attributes that define what has been done during the change implementation, the traceability would be much more effective.

4.4 Version management

Most SCM tools provide support for configuration identification and version control, allowing software development to be integrated directly with SCM procedures. Naturally not just code but all items capable of being stored electronically may be version controlled by SCM tools. For effective configuration management operations, a version control system must meet, as a minimum, the following criteria [14]:

- Manage access to, and control the sharing of, sources in multi-user parallel development environments, teams must have visibility of each other's changes, and access restrictions are essential to prevent unauthorized access, access at inappropriate times and multiple accesses.
- Support parallel development, allowing easy creation of CI variants and, if appropriate, their later reintegration or merging — this support is essential where

components are being developed by several people at the same time, or if bug fixes are required to maintain older released components.

- Store change histories — records must be kept of all changes applied to all CIs. The change history should show how and why a CI has evolved over time. It provides an audit trail, essential for all but the most trivial of projects.
- Allow the creation and management of baselines.

In our proposed model, the parallel development mechanism used is the "Copy-Modify-Merge" in which the developers are allowed to work on the same file simultaneously with no exclusive locks on the file. The SCM tool should be able to assume the burden of integrating all the changes, and keeps track of any conflicts.

Visual differences should also be provided to illustrate the conflicts between two or more developers work, also in case of a failure in merging the conflicts between the developers, the SCM tool should be able to retrieve the original version before the developers started changing the document.

This can be done by keeping a local copy in each developers workspace alongside server up-to-date copies (with temporary tag on them) to retrieve the copies in case of a developers workstation crash. Also the original file is still in place on the server and is still considered the latest version (the working one) until a successful check-in of the developers in progress copies takes place and succeeds it (which will create a new version for the file in the SCM repository).

The problem with file visual differencing in most of the version mechanisms is that it does depend on one and only one methodology for conflict resolution, regardless of the file type. In each version control software, a mechanism is defined for differencing between files, for example some tools use the character-by-character differencing to differentiate between files, this technique is so thorough and in-depth. While some other tools use the file-by-file differencing, which is generic and performance-focused. Most of the tools use the line-by-line technique, which is moderate and widely-accepted. But in our proposed model, we have developed a mechanism for visual differencing to consider the file type in hand.

Since documents tend to have lots of lines in it, the proposed model would suggest a line-by-line approach to deal with the conflicts between developers. But if a binary file was the subject of the conflict resolution process, the proposed model would suggest the file-by-file approach to deal with the conflict, since it does not make sense to differentiate between two binary files including hundreds of binary code using the line-by-line technique.

And in case the file in hand was an image file, the character-by-character technique would be suggested by the proposed model, since each pixel does matter in an image. The dynamicity that the proposed model provides in conflict resolution, would save a lot when it comes to storage space and performance perspectives, dealing with each file in the most suitable methodology

would come very handy also for the developers trying to work concurrently on a project since they know they have no real issues in resolving any conflicts when the time for check-ins comes.

A noticeable shortage in most of the version management modules discussed earlier is the lack of support for obvious bug fixing operations, as introduced later in this chapter. In our proposed model, we gave the flexibility for the management of the project to include an extra form beside the regular Change Request (CR) form, which is called "PR" which stands for Problem Report. The proposed model differentiates between a change request and a problem report and that can be used in version management module.

If the form that initiated the change task in hand is a PR, then the affected CIs will not have new versions defined, which makes sense because there is no need to define a new version each time a bug is fixed, this could lead to large number of insignificant versions that have bugs and are not useable in any configuration build, and would contradict the meaning of versioning since each version differ from the successor of it in functionalities.

The PR differs from the CR in many ways, first of all, there is no retrieval of the previous version of the file once the PR is committed and approved, since the older version would be totally replaced, and in the tracking database a record of who initiated the PR and who implemented it is recorded for issue tracking.

Also the version would not change or increase once the file has been submitted using a PR form, for those reasons, we limited the PR privilege to be only for the team leader, who could initiate the PR and assign it to a developer to implement. And hence we have control over the PR process in a way that will make lighter in means of work flow than the CR, but also well-controlled since the initiator of the PR is a team leader.

Also another limitation in several versions management modules is that it does not allow a developer to view a read only copy of a certain file unless a check-out command is issued, this can be very frustrating for software developers since the reason behind his well to view that file could be for obtaining information from the file for another set of files development, and not to change the file itself.

In our proposed model, the workaround for this issue is a defined command which will open a read-only file on the client's workstation, and will not create a local copy of the file on his own workspace. And that is accomplished using the temporary folder of the operating system or a predefined temporary folder for the system on each developer's workstation as per the administrator's installation.

The read-only command would be recorded in the transactions database, and would be an alternative solution for the developers than using check-outs for retrieving a viewable file. Also the proposed model notifies the developer if the file issued in the read-only command was checked out by other users. The purpose behind this notification is that the developer reading the

file should know that ongoing editing on the file is in progress, and that a new version of that file might be submitted soon. This piece of information would come very handy for the developers integrating files from different modules in the project to each other especially in an environment where each developer works on a subsystem of the whole system, and does not have any idea of what is going on in other subsystems.

4.5 Change Control

Version control is often regarded as the traditional flavor of software configuration management. While it is fundamental, alone it is certainly not sufficient. In the development of any product, but especially for software systems, the only guarantee is that the system will evolve from its initial specification, and that problems will be found throughout its life cycle. Change control is the process whereby changes arising through a product's life cycle, whether enhancements or bug fixes, are initiated, progressed and managed.

In the subsections below, we will redesign the change lifecycle in a way that could benefit greatly in time and cost savings. The main purpose behind redesigning the change lifecycle is that SCM tools tend to treat all kinds of changes in the same methodology. Neglecting the change type could cause lots of overhead in change tasks. .

4.5.1 Change Request Forms

In the previous change model, the initial form is called a "CR" which represented simply a change desired in the system functionality, no matter what the change type was, this form was mainly used. But in our proposed model; three different kinds of forms to initiate the change lifecycle are introduced.

The first and most simple form is (PR) which stands for Problem Report. PRs are for reporting problems that were raised mainly by system testers or developers for troubleshooting purposes, such as specification non-compliances, bugs, performance and usability issues, etc. those PRs are initiated by a team leader or someone who has the privilege to initiate a PR as per the administrator permissions set, and then assigned to a developer to be implemented. The PR form is illustrated below.

Index	Field	Description
0.0	PR identification	This section identifies the PR itself, a unique name or sometimes a software-generated unique

		identifier is attached
1.0	CI(s) identification	This section identifies the CI(s) for which a bug fix has been requested. Those CIs are the objects that the bug fix will mainly be implemented to.
1.1	CI(s) identifiers	The name, version management unique identifiers numbers of the CI(s) is specified, including page numbers if a document is involved. Also a brief description is provided.
1.2	PR Requester	The name of the person or the group requesting the bug fix.
1.3	Bug fix date, location, and time	When and where the bug fix was requested
2.0	Description of the bug fix	This section presents a brief description of the requested fix
2.1	The bug fix description	A detailed discussion of the bug fix requested and supporting information if existed.
2.2	Requester priority	The priority assigned by the requester to the work generally chosen from a four-value scale. Critical, High, Medium and Low are the scale values.
2.3	Attached documents	If any attached documents are included in the bug fix request, it is referenced here.
3.0	Manager decision and review	Describes the manager's decision about the PR, and his review.
3.1	Manager Decision	This section describes if the PR was approved or rejected (the below sections will be filled only if the PR was approved)
3.2	Manager action description	This section describes the actions for the bug fix implementation, the persons in charge of the implementation
3.3	Manager priority	PR priority according to the Manager. The priority four-value scale is also considered here
3.4	Estimated Completion Date	When the PR is expected to be finished

Table 4.1: PR Form

The second type of forms is the Change Request (CR). CRs are for functional enhancements that were raised by a developer, customer, quality assurance officer or the project manager to enhance the software productivity, performance, or simply adding a noticed missing feature. The CR form is illustrated below.

Index	Field	Description
0.0	CR identification	This section identifies the CR itself, a unique name or sometimes a software-generated unique identifier is attached

1.0	CI(s) identification	This section identifies the CI(s) for which a change has been requested. Those CIs are the objects that the change will mainly be implemented to.
1.1	CI(s) identifiers	The name, version management unique identifiers numbers of the CI(s) is specified, including page numbers if a document is involved. Also a brief description is provided.
1.2	Change Requester	The name of the person or the group requesting the change.
1.3	Change Date, location, and time	When and where the change was requested
2.0	Description of the change	This section presents a brief description of the requested change
2.1	The change description	A detailed discussion of the change requested, its motivations and supporting information if existed.
2.2	Reasons and justification	This section discusses why the change has been requested and the justification for the request.
2.3	Perceived affects	The requester's perception of the affects of the change.
2.4	Requester priority	The priority assigned by the requester to the work generally chosen from a four-value scale. Critical, High, Medium and Low are the scale values.
2.5	Attached documents	If any attached documents are included in the change request, it is referenced here.
3.0	Manager decision and review	Describes the manager's decision about the CR, and his review.
3.1	Manager Decision	This section describes if the CR was approved or rejected (the below sections will be filled only if the CR was approved)
3.2	Manager action description	This section describes the actions for the change implementation, the persons in charge of the implementation, and what is expected from the change
3.3	Manager priority	CR priority according to the Manager. The priority four-value scale is also considered here
3.4	Manager estimated cost	This section prepares the project manager for what is expected to cost through undergoing the change. Labor and time costs are included often here in this section.
3.5	Estimated Completion Date	When the CR is expected to be finished and ready for integration.

Table 4.2: CR Form

The third type of forms is the Major Change Request (MCR). MCRs represent a major change in the system design, architecture, blackbox components or the application database. Those types of changes require special treatment and sophisticated approval cycles to gain better traceability of the change in hand. The MCR form is illustrated below.

Index	Field	Description
0.0	MCR identification	This section identifies the MCR itself, a unique name or sometimes a software-generated unique identifier is attached
1.0	CI(s) identification	This section identifies the CI(s) for which a change has been requested. Those CIs are the objects that the change will mainly be implemented to.
1.1	CI(s) identifiers	The name, version management unique identifiers numbers of the CI(s) is specified, including page numbers if a document is involved. Also a brief description is provided.
1.2	Change Requester	The name of the person or the group requesting the change.
1.3	Contact information	How to contact the requester.
1.4	Change Date, location, and time	When and where the change was requested
2.0	Description of the change	This section presents detailed description of the requested change
2.1	Description	This section presents a detailed description of the circumstances and motivations that participated in the change request desired.
2.2.1	Underlying circumstances	Background information regarding the request.
2.2.2	Examples	Supporting information, e.g., printouts containing an error in a report or an incorrect screen image or a system deliverable that contradicts the business objectives
2.2.3	The change	A detailed discussion of the change requested.
2.2	Reasons and justification	This section discusses why the change has been requested and the justification for the request.
2.3	Perceived affects	The requester's perception of the affects of the change.
2.4	Alternatives	The requester's perception of any alternatives to the change.
2.5	Requester priority	The priority assigned by the requester to the work generally chosen from a four-value scale. Critical, High, Medium and Low are the scale values.
2.6	Attached documents	If any attached documents are included in the change request, it is referenced here.

3.0	CCB decision and review	Describes the Change Control Board (CCB) decision about the MCR, and their review. Some organizations separate this section through a dedicated form.
3.1	CCB Decision	This section describes if the MCR was approved or rejected (the below sections will be filled only if the MCR was approved)
3.2	CCB Review and Justification	This section describes the understanding of the committee for the MCR, and the justification behind the decision
3.3	CCB action description	This section describes the actions for the change implementation, the persons in charge of the implementation, and what is expected from the change
3.4	CCB priority	MCR priority according to the CCB committee. The priority four-value scale is also considered here
3.5	CCB estimated cost	This section prepares the project manager for what is expected to cost through undergoing the change. Labor and time costs are included often here in this section.
3.6	Estimated Completion Date	When the MCR is expected to be finished and ready for integration.

Table 4.3: MCR Form

Those forms differ from each other in the phases required to accomplish them in our proposed model, and in the form fields too.

4.5.2 Change Life Cycle

As illustrated earlier in chapter 2, a change life cycle begins with the identification of an existing need for a change in the course of the software development, and then an assigned developer may provide a preliminary evaluation of the change request, a missing part in that cycle was the iteration of the evaluation itself, the evaluation itself might need an evaluation from the members of CCB, and if missing or vague information was discovered in the evaluation, the form must be commented on and returned to the developer who ran the initial evaluation for enhancements.

Depending on the change type being processed, the company must be able to define its own customized steps in the change lifecycle to deal with the request in hand. by that we mean that whenever a file or a set of files are subject to change, the lifecycle should be shortened or lengthened according to the change type and its sensitivity, also the people reviewing the change should not be the same for each type of change, and hence depending on the change type, the

approval and workflow cycle would be chosen, and the tool being used should be able to support those set of customized rules to be placed in the change lifecycle.

After change requests and problem reports are raised they pass through a number of defined life cycle phases. The proposed model takes into consideration that the number of different life cycle models through which the forms could progress should be kept at minimum to increase productivity.

The generic life cycle model offered for project support Configuration Management Unit is illustrated in Figure 4.2.

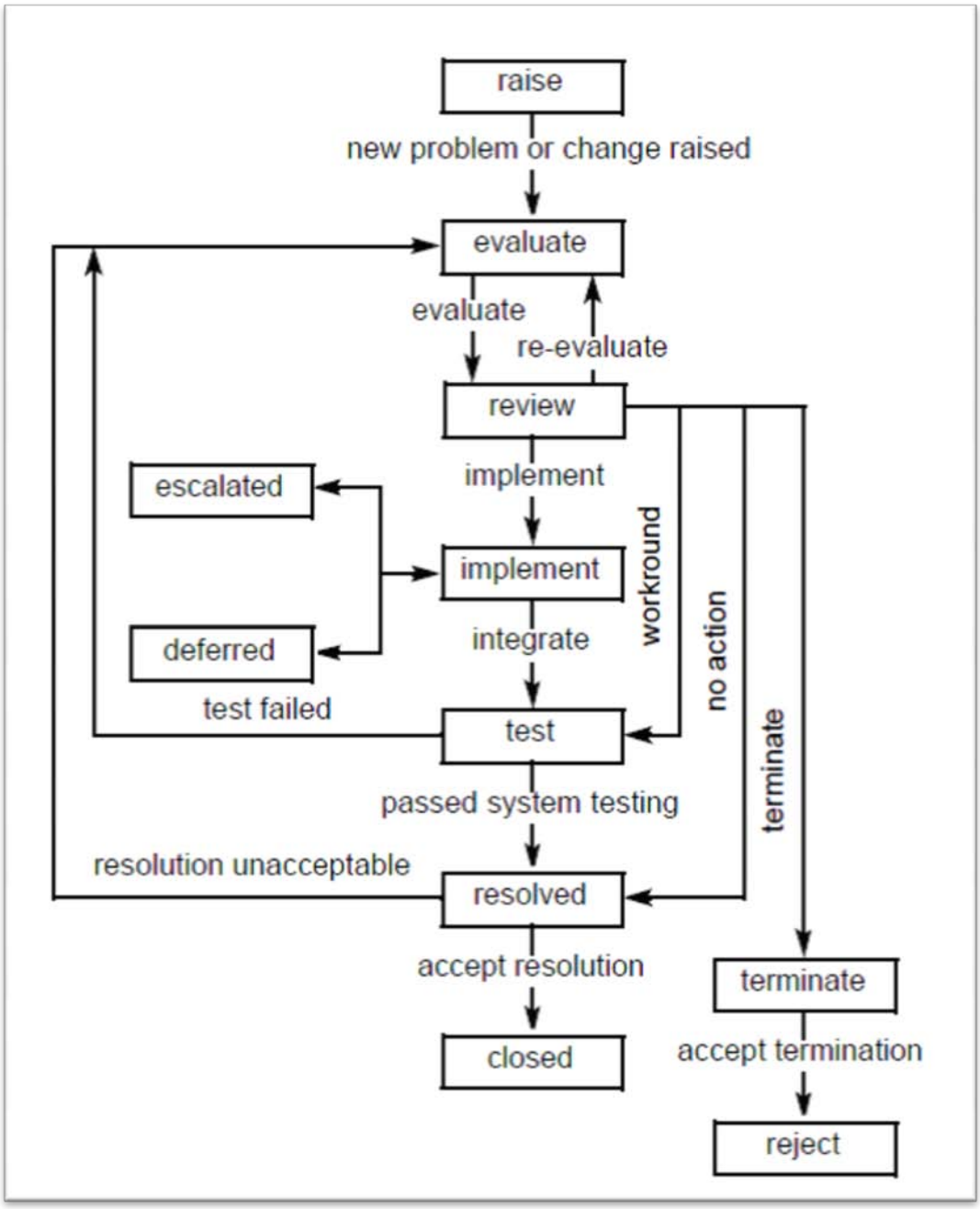


Fig 4.2: A typical change request life cycle [29]

After the request is raised, a technical evaluation is performed, with the aim of identifying the problem and the CIs responsible. It is then reviewed for severity, priority and impact upon the project by a Change Control Board (CCB). It is the CCB's responsibility to approve, monitor and control the conversion of design objects into system CIs and authorize changes to that system. [29]

At this stage in the change life cycle the CCB may [29]:

- Decide to implement the request, and scope its inclusion against a release,
- Request further technical evaluation,
- Perform more detailed impact analysis, of both implementing and not implementing the change,
- Reject the request — for example a CR outside the scope of the product's requirements,
- Provide a work-round not requiring any changes.

Those requests having a serious impact on the product or customer may cause the change to be escalated, while low-priority changes may be deferred to later releases. Changes then follow the implement, unit-test, integrate, integration-test cycle until they can be closed. If implementation is not successful, the request may need reevaluation, for example, a fault cause may not have been where expected. [29]

Most of the SCM tools follow the change lifecycle illustrated above. The problem with that lifecycle is that it does not take into account the type of change at hand (bug fixing or enhancement), and does tend to apply the same lifecycle to all types of change.

In the following figure 4.3 is a more detailed lifecycle for the change request in our proposed model, the details satisfy most change types, and set ground rules for any company looking to derive its own customized lifecycle.

In our proposed model, the experience of the developer using the lifecycle begins with the log in process that will notify all other online developers that a certain developer has joined the project, then the master plan of the project appears to the developer enabling him to choose from the tasks that he was assigned to by the project manager or the team leader.

Our proposed model uses a hybrid approach between lock-modify-unlock and copy-modify-merge methodologies (as will be illustrated later). We used each methodology to satisfy certain change type.

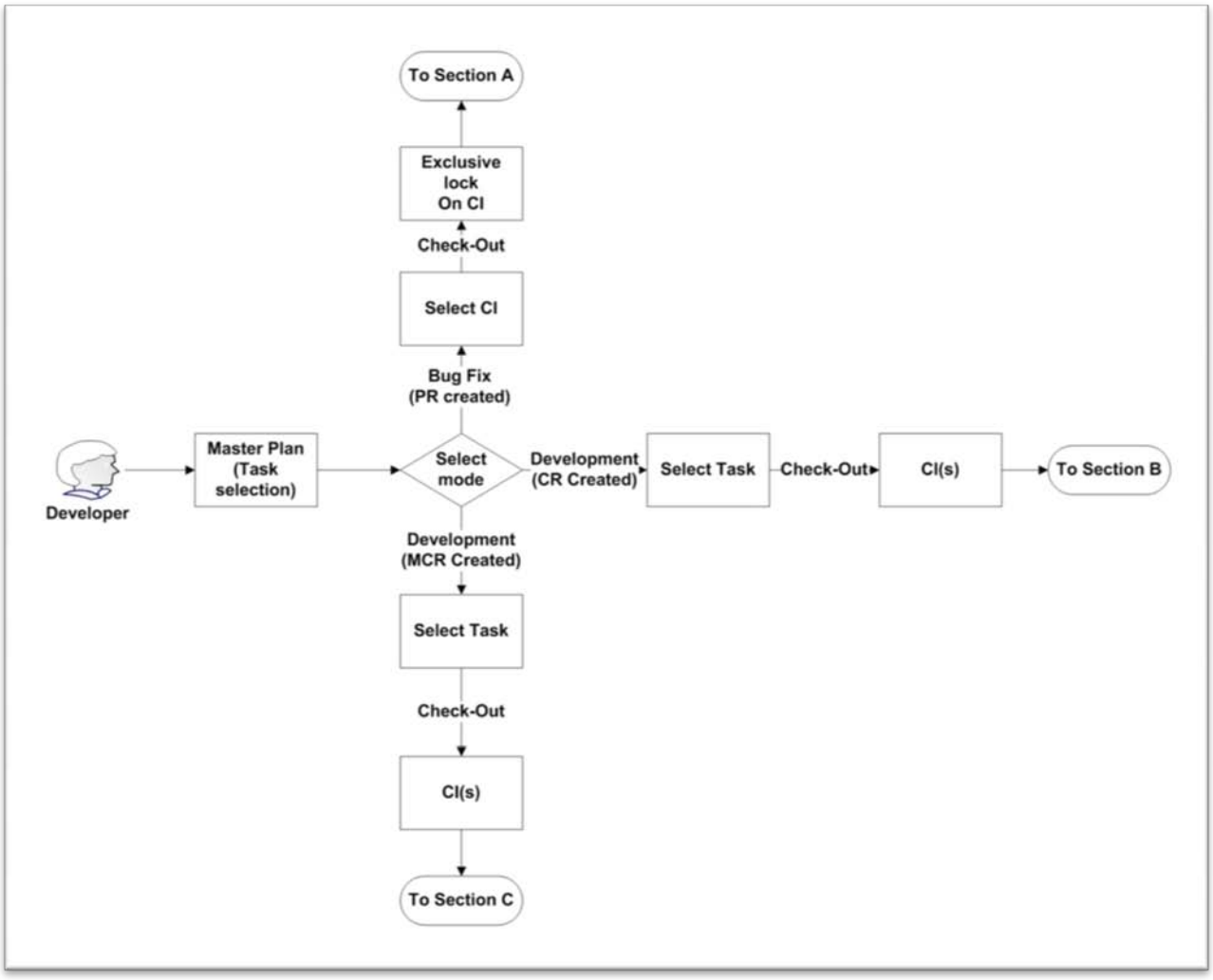


Figure 4.3: Proposed change lifecycle

After the user selects a task to work on, he'll have to choose between three different modes of operation, development (CR and MCR) or bug fix (if any PRs were assigned to him). If the purpose was to continue the development on the task that was selected from the master plan, then the developer will have to select certain CIs to develop, and the SCM will manage the proceeding check-out operations on the related CIs.

In case a bug fix was assigned to the developer, the model would adapt to bug fixing as illustrated in figure 4.4.

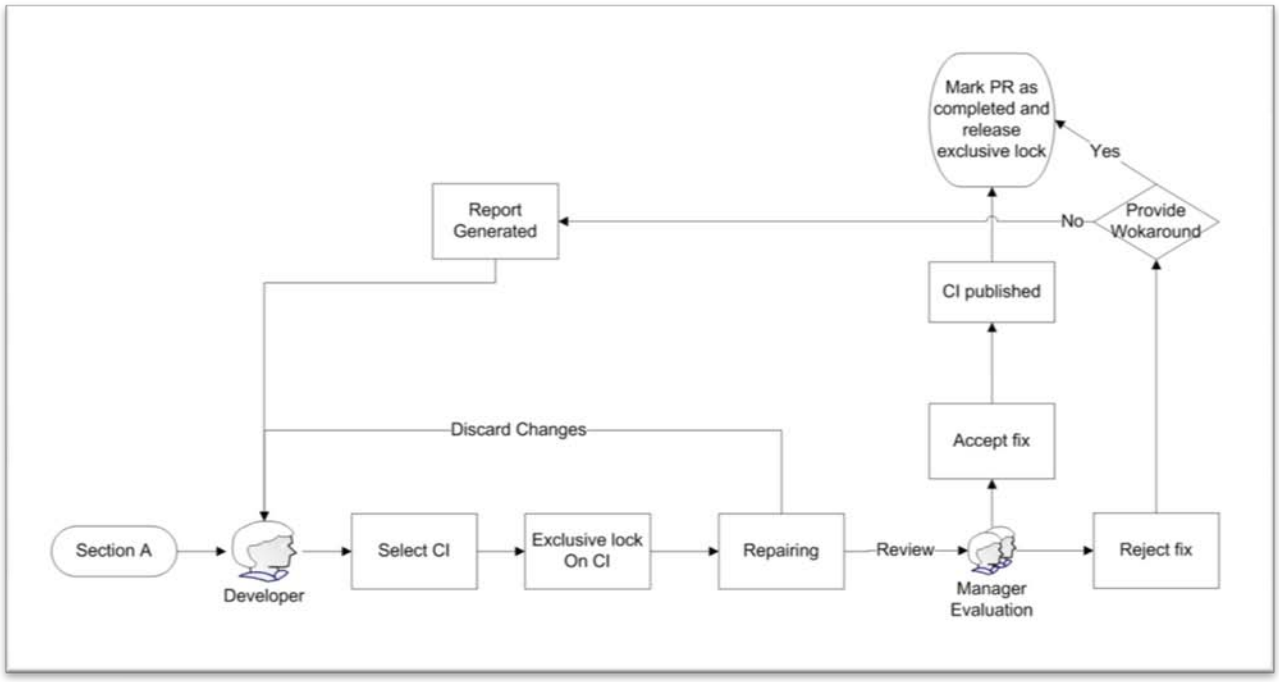


Figure 4.4: Bug fixing mode in our proposed model

The developer will acquire exclusive locks in the intended CIs, since the mode will fix errors and malfunctions in the CI, no other users would have access to that CI until errors are resolved. This type of change usually takes little amount of time compared to other changes, hence exclusive locks would not limit parallel development for a long time.

Then the developer would start working on the CI, repairing the errors in it, then submitting it back for reviewing by his manager. If approved, the PR would be marked as done, but if not, the manager would provide a report explaining the reasons behind his decision, or he could provide a workaround.

If the developer has development tasks to work on, he would rather select CR, or MCR as illustrated in figure 4.3 above.

If the developer selects a CR to implement, the model would adapt to CR mode as illustrated in figure 4.5 below.

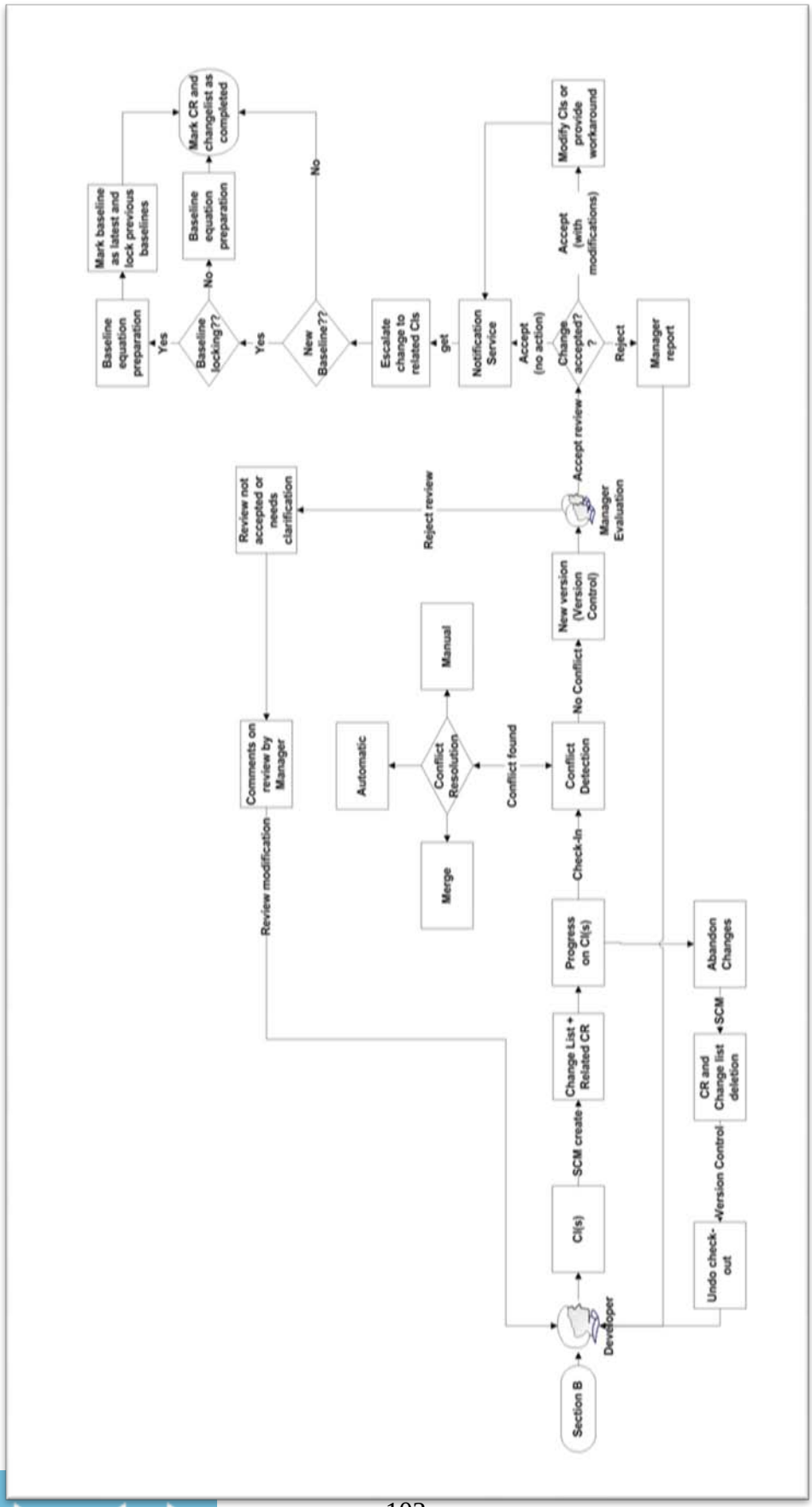


Figure 4.5: CR mode in our proposed model

The SCM will create a change request to document the procedure, and will assign the CIs to the change request, and will relate the change request to the task selected by the user from the master plan in the first step. Then the developer will have his own copy in his workspace on his workstation prepared by the SCM system.

After the needed changes have been completed by the developer, comes the time for submitting the changes back to the centralized repository. The version control mechanisms here takes into consideration all developers who have a check-out marker on related CIs, and then will detect any conflict between the developers' changes, and if conflicts were found, three methods would be available for conflict resolution supported by visual differencing that was introduced earlier in this chapter.

After the conflicts were resolved or simply no conflicts were found, the CIs are assigned new versions through simply increasing the version number by 0.1, or by asking the user to input his own version number, and then those new versions are submitted to the change control board to evaluate and take a decision regarding whether to accept it or not.

When a developer changes a set of CIs, he needs to provide a review report to clarify what kind of effort was introduced to the related components, and what kind of tests were established to assure that changes really work, that report alongside the changed CIs are passed to the change control board to evaluate. Here the board can ask for a re-make of the after-change report in case of missing or vague information within the report itself. But if the report was sound and clear and accepted, the CIs themselves are then examined against the report results.

Here the manager can either accept the changes as were implemented by the developer, reject those changes for good or accept the changes alongside some modifications (workaround).

In case the changes were not accepted, the manager should provide a report describing the reasons why the change was dismissed, and then retrieve the project to the previous baseline by discarding the changelist and CR, and rolling back checkouts.

And if the changes were accepted, either completely or partially (with a workaround provided), the notification service would request names and lists of related people to notify, the SCM can retrieve those lists from the change requests that are active and have a related CI to those that were changed in it, then the notifications service notifies the appropriate set of people either through emails, instant messaging or SMS, depending on the administrator's installation.

The next step is to escalate changes to related and affected CIs, and when the project manager receives a notification of the change, he checks if the change is major and assists him in that decision is the report that was generated from the manager review, and if he decides that it is an

important change in the project's lifecycle, a new baseline is set and defined, and this baseline could be either locked (as discussed earlier in the baselines section) to prevent further changes to previous CIs, or could be left as is.

If the change at hand was a major change, an MCR would be created, the MCR has more fields and provides much more detailed information for the change evaluation and implementation. The model would adapt to MCR mode as illustrated in figure 4.6 below.

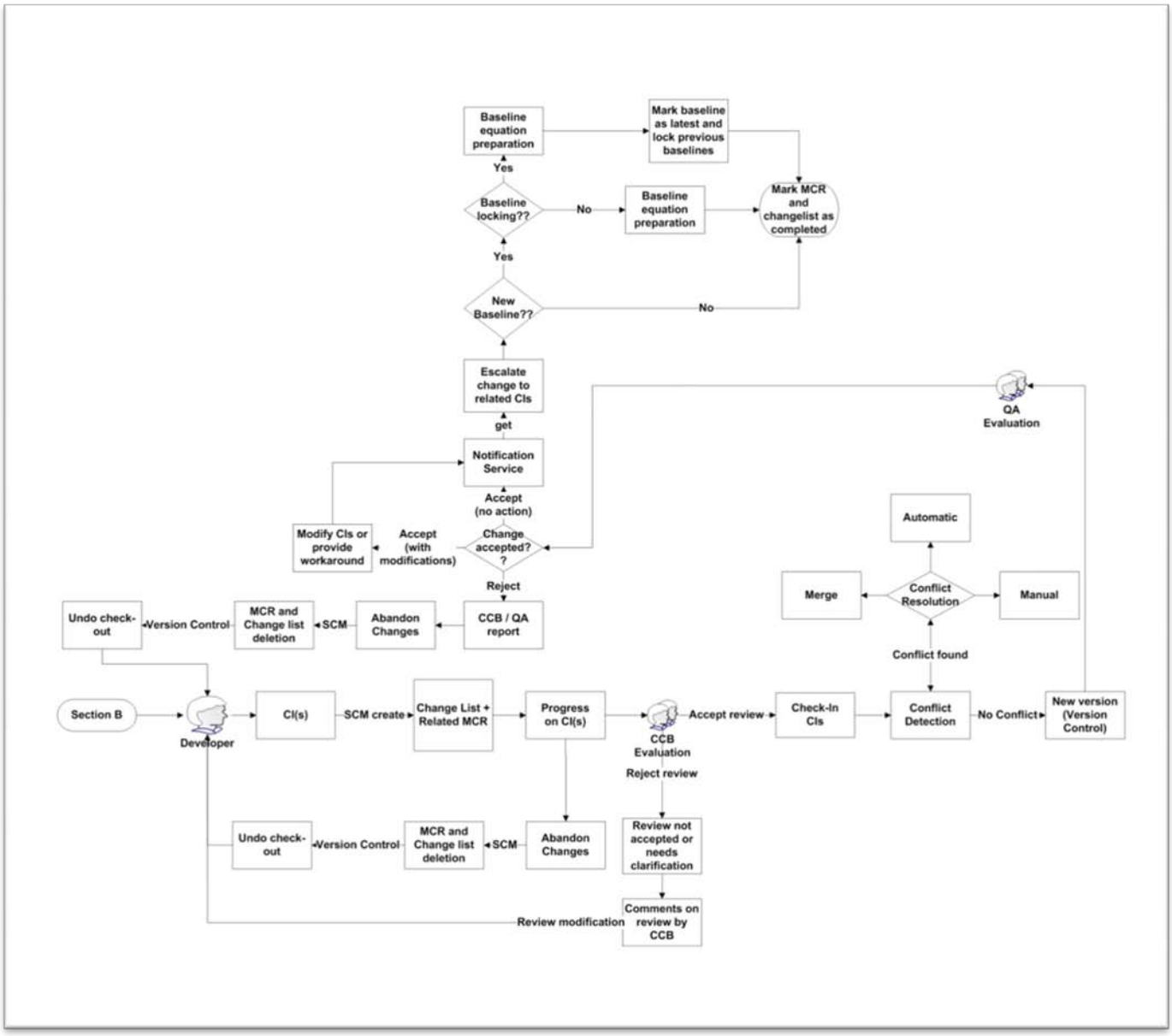


Figure 4.6: MCR mode in our proposed model

Our proposed model differentiates between CR and MCR mostly in the evaluation and approval phases, which are the turning points in the change lifecycle. The CR is reviewed and approved by the developer's manager.

But MCR is approved by the CCB before the change is really reflected in the project repository, because of the sensitivity of the change itself and its broad effect. After the change has been reflected in the project's repository, the Quality Assurance department reviews the results and integration and provides their feedback. Also QA has the power to abandon the changes and reverse the whole operation.

The main advantage behind our proposed change lifecycle is that we linked the change requests and problem reports to CIs in a more productive methodology. The proposed model closely integrates the disciplines of version control and change control. The first reason is to relate CI changes to CRs and PRs, and the second is to enforce the version management system access controls.

As anyone could notice through the description of the proposed model change lifecycle, it is far more sophisticated than the original lifecycle, and takes into consideration users' roles in the project and the type of change at hand, which can be very useful in assisting the company spread the tasks across its various types of employees more efficiently.

Also the proposed model being built on task-based basis gives the middle and upper management in the project hierarchy more visibility when dealing with changes. Since they have each CR connected to a task predefined in the project main blueprint, and those set of tasks connected to baselines in a more meaningful way. The main purpose behind the redesigning of the change lifecycle was to give the set of change transactions more managerial perspective and connecting them to be more productive than being just a transaction with description.

Chapter 5

Conclusions and Future Works

5.1 Conclusions

Now that we have introduced the nature of local market in the field of SCM implementation, and that we have mentioned aspects and challenges facing the local organizations in implementing SCM, and how SCM is exactly introduced into a software development framework, and how this framework would be changed in case of the SCM presence, we can conclude the fact that implanting our proposed model in a local market organization's development framework would be hard to achieve, especially that it is only theoretical tool and has not been implemented.

Since local market organizations tend to have alternative ways of handling the change process, the proposed model modifications to the standard SCM lifecycle that we have discussed in chapter 4, will have no effect on the environment intended for implementation because the original SCM lifecycle that was the base of the proposed modifications is not present in those organizations. And hence we were unable to test our proposed model in any local organization because of the lack of proper testing environment.

But to evaluate our proposed model, we designed a questionnaire and distributed it to multiple software development companies in the Jordanian market. We included in the appendix the questionnaires filled by the organizations that were interested in the subject and offered their help and time in discussing their environments and evaluated our proposed model.

The selected sample companies have experience in the software engineering practices, especially in the software configuration management field. The companies varied in size and the employees interviewed varied in their job positions.

So we designed the questionnaire to accomplish multiple goals. These goals are mainly:

1. Get familiar with their working environment in the fields of project sizes, number of developers, project hierarchy, etc…
2. How local organizations tend to manage changes during their projects
3. The proposed model was presented to the person interviewed, then his evaluation of the proposed approaches was recorded

The interviews made the basis for the local market results that were presented earlier in chapter 5, and revealed the challenges that those organizations tend to face in managing change in their development projects.

From the interviews we conclude that the model adds sophistication and flexibility in major SCM fields. Table 5.1 below represents statistics about each SCM field and its relative results in the interviews.

Proposed Model Component	Who said it is useful	Who said it is irrelevant	Who said it is useless
Baselines Presentation	A – B – C – D – E - F		
Version Management	C – D - E - F	A - B	
Change Lifecycle	B – C – D - E	A - F	

Table 5.1: Interviews statistics and results

As illustrated in the table above, most of the software development companies appreciate the additions and alternative methodologies offered by our proposed model. Six out of six companies found the baselines redesign helpful. And four out of six companies found the version management approach in our proposed model helpful. And in the field of the proposed change management lifecycle, four out of six companies found the proposed change lifecycle helpful.

Also multiple companies showed interest in the chapter 5 sections which discussed SCM introduction into a software development environment. They commented that "challenges in implementing SCM" section prepares the company intending to implement SCM for those challenges and ease the transformation process.

In Table 5.2 illustrated below, are the answers for all interviews grouped together to gain better analysis of the results found during those interviews.

Questions	Interview A	Interview B	Interview C	Interview D	Interview E	Interview F
Overall developers	17	20	100	800	50	20
Project typical team	6	20	50	400	15	10
User roles in a project	PM, TL, SD, JD	PM, TL, SD	PM, TL, SD, TM, PD	Many roles	PM, BA, TL, SD, JD	PM, PC, BA, SD
Change management procedures	Minor change handled by TM, major change handled by PM	QA initiates the CR, PM makes a decision	Major change handled during project meetings	Major changes approved by CCB	Minor change handled by TM, major change handled by PM	Customer request through CR, development team evaluates
CCB committee presence	No	No	No	Yes	Yes	No
Formal change request	No	Yes	Yes	Yes	Yes	Yes
VCS tool in the environment	SVN- VSS-TFS	TFS	TFS	SVN	VSS – TFS	VSS
SCM tool in the environment	TFS	TFS	TFS	ClearCase	TFS	No
VCS functionalities in the environment	Versioning basics	Versioning basics	Artifacts storage, versioning, branching	Parallel development, labeling baselines	Versioning basics	Access permissions, storage of CIs
SCM functionalities in the environment	Versioning, release management	Versioning basics	Checkout handling for artifacts (exclusive locks)	Managing CIs, change control, change record and audit	Versioning, storage, tasks management, document management	No SCM tool
Who accesses the SCM tool	Team members	Development team	All users	Team members	Team members	No SCM tool
Conflict resolution methodologies	Exclusive locks, no conflicts	Exclusive locks, no conflicts	Exclusive locks, no conflicts	Exclusive locks, no conflicts	Manual conflict resolution	Manual conflict resolution
Proposed baseline approach	Useful	Useful	Useful	Useful	Useful	Useful
Proposed version management approach	Irrelevant to business	Irrelevant to business	Useful	Useful	Useful	Useful
Proposed change lifecycle	Irrelevant to business	Useful	Useful	Useful	Useful	Irrelevant to business

Table 5.2: Interviews analysis

The legend of the table above is mentioned below:

- PM: Project Manager
- TL: Team Leader
- BA: Business Analyst
- SD: Software Developer
- JD: Junior Developer
- TM: Technical Manager
- PD: Project Director
- PC: Project Consultant
- VSS: Visual Source Safe
- TFS: Team Foundation Server
- SVN: Subversion

5.2 Future Works

We would like to suggest some interesting issues and ideas that could not be reached because of the time, resources and other constraints and they will aid as an improvement on the proposed model. As future work, we mention:

- Full implementation of the proposed model for Software Configuration Management Ultimate Tool. This can be done by a selecting software development vendor to implement our proposed design of the tool, implementing the suggested model and finally applying an experimental test by allowing a group of software engineers to try out the system and assess the software configuration experience and the effectiveness of the proposed model in fulfilling their desires.
- Enhancing the build, release and branching methodologies embedded in SCM tools to make them more appealing and productive (our interviews indicated that most companies rely on specialized tools to perform build and release management)
- Adjusting the suggested model and enhancing its reliability in the field of offline batches; this will become very handy in software development environments with geographically dispersed developers.
- Studying the influence of social matters in the implementation of the SCM tools in local market, and providing solutions and investment encouragements for organizations interested in the concept of implementing SCM lifecycle in their environments.

References:

[1] Bach J. (1998), "The Highs and Lows of Change Control," *Computer*, vol. 31.

[2] Bar M., Fogel K., (2003) Open Source Development with CVS, 3RD EDITION. Paraglyph Press, Inc.

[3] Beck J. (2005), Using the cvs version management system in a software engineering course.

[4] Ben-Menachem M. (1994), Software Configuration Guidebook (McGraw -Hill International Software Quality Assurance Series), McGraw-Hill, October 1994.

[5] Berliner B. (1990), CVSII: Parallelizing Software Development, in proceedings of USENIX Winter 1990 Conference, Washington D.C.

[6] Bersoff E., Henderson V., Siegel S., (1980), Software Configuration Management, An Investment in Product Integrity.

[7] Clemm G. (1989), "Replacing Version Control with Job Control," Proc. 2nd Intl. Workshop on Software Configuration Management, ACM, Princeton, NJ, october 1989.

[8] Collins-Sussman B. (2002), The Subversion Project: Building a Better CVS, Linux Journal, Feb 2002.

[9] Collins-Sussman B., Fitzpatrick B., Pilato C. (2009), Version Control with Subversion for Subversion 1.6.

[10] Conradi R. (1998), Version Models for Software Configuration Management. ACM computer Surveys (CSUR), Volume 30, Issue 2, Pages: 232-282, ACM press, June 1998.

[11] Dart S. (1990), Spectrum of Functionality in Configuration Management Systems. Carnegie Mellon University

[12] Dart S. (1991), Concepts in Configuration Management Systems. U.S. Department of Defense

[13] El-Khalili N. and Damen D., (2005) Software Engineering Practices in Jordan.

[14] Gouker T. (2009), Information Technology-Enabled Change Management: An Investigation of Individual Experiences

[15] IEEE Guide to Software Configuration Management, ANSI/IEEE Std. 1042-1987. (1987)

[16] IEEE-Std-610 (revision and redesignation of IEEE-Std-729-1983), (1990).

[17] Khanna S., Kunal K., and Pierce B., (2002) A Formal Investigation of Diff3.

[18] Leisten S. (2007), Evaluating a CM System for Agile Development, Version 1.4.

[19] Nagel W. (2005) Subversion Version Control. Prentice Hall, Professional Technical Reference, ISBN 0-13-185518-2

[20] Nguyen T. (2005), Object-oriented software Configuration management, University of Wisconsin-Milwaukee, August 2005.

[21] Nuraminah R., Shukor Sanim M., Nasir H. (2009), the Development of Software Configuration Management Repository, International Conference on Information Management and Engineering.

[22] Perforce 2010.1 Introducing Perforce, June 2010, Perforce Software. (2010)

[23] Pressman R. (2005), Software Engineering a practitioner's approach, Sixth edition. McGraw-Hill
[24] Rational Claercase Introduction, Rational the software development company, http://www.rational.com, (2003)
[25] Robbes R. & Lanza M. (2005), Versioning Systems for Evolution Research
[26] Ruparelia N., (2010), The History of Version Control. ACM SIGSOFT Software Engineering Notes, January 2010 Volume 35 Number 1.
[27] Sommerville I. (2007), Software Engineering, Eighth edition. McGraw-Hill
[28] Subversion: Not Just for Code Anymore.
[29] Thompson S. (1997), Configuration management — keeping it all together, BT Technol J Vol 15 No 3 July 1997
[30] Tichy W. (1985), RCS—A System for Version Control.
[31] TIMe electronic textbook (1990), Configuration management, (1990)
[32] Using Unified Change Management with Rational Suite, Rational Software Corporation, (2003)
[33] Voinea L. & Telea A.(2006), An Open Framework for CVS Repository Querying, Analysis and Visualization, MSR'06 2006 ACM 1-59593-085-X/06/0005.
[34] Wahli U., Brown J., Teinonen M., Trulsson L. (2004), Software Configuration Management A Clear Case for IBM Rational ClearCase and ClearQuest UCM.
[35] Wasson J. (2009), Configuration Management for the 21st Century, White Paper.
[36] White B. (2000), Software Configuration Management Strategies and Rational, CleaseCase: A Practice Introduction.
[37] Whitgift D. (1991), Methods and Tools for Software Configuration Management.
[38] Zhang Y. (2005), software configuration management, dalhousie university Halifax, Nova Scotia August 2005

Web References

[39] Apache Subversion. http://svn.apache.org, last visited September 2, 2010.
[40] Burlington Telecom, http://www.burlingtontelecom.net/~ashawley/rcs/manual/html/ch07s04.html A Manual to the GNU Revision Control System (RCS), Last visited January 2, 2011.
[41] Internet FAQ Archives. http://www.faqs.org/faqs/sw-config-mgmt/cm-tools/, Last visited July 23, 2010.
[42] Linux Journal. http://www.linuxjournal.com/article/2378, Last visited September 2, 2010
[43] Massachusetts Institute of Technology. http://web.mit.edu/ghudson/info/fsfs, last visited September 2, 2010.
[44] Perforce Software. http://www.perforce.com/, Last visited September 2, 2010 .

[45] Producing Open Source Software, http://producingoss.com/en/vc.html, Last visited January 2, 2011.
[46] Red Bean for CVS, http://cvsbook.red-bean.com/cvsbook.html Last visited July 23, 2010.
[47] SGI.http://techpubs.sgi.com/library/dynaweb_docs/0620/SGI_EndUser/books/ClrC_UG/sgi_html/ch01.html, Last visited September 2, 2010 .
[48] Slide Share, http://www.slideshare.net/maheshpanchal1/software-configuratiom-management-role-n-resposnbilities Last visited January 2, 2011.
[49] Wikipedia.org. http://en.wikipedia.org/wiki/Apache_Subversion, Last visited September 2, 2010.
[50] Wikipedia.org. http://en.wikipedia.org/wiki/Clearcase, last visited September 2, 2010.
[51] Wikipedia.org. http://en.wikipedia.org/wiki/IBM_Rational_ClearCase_UCM, Last visited September 2, 2010.
[52] Wikipedia.org. http://en.wikipedia.org/wiki/Perforce, Last visited September 2, 2010.
[53] Wikipedia.org. http://en.wikipedia.org/wiki/Product_manager, Last visited November 25, 2010.
[54] Wikipedia.org. http://en.wikipedia.org/wiki/Revision_Control_System, last visited July 23, 2010.
[55] Wikipedia.org. http://en.wikipedia.org/wiki/Software_configuration_management, Last visited September 2, 2010.

Appendices

Appendix A: Formal Interviews

Interview A:

Person Name: Khair Al Arda

Person Position: Project Manager

Company Name: ExcellentSoft

How do you describe the projects size developed by your organization?

Medium to large

How many developers are there in the company?

12 full time + 5 outsourcing

How many developers work on a typical project (average)?

6

What user roles are available in the project hierarchy?

Project manager, team leader, senior developer, junior developer

How does the project team handle the process of change?

Minor change will be evaluated verbally, major change the manager will issue a change of cost and time to the customer to evaluate and then proceed, of course it takes a different approach if the request if adding feature or updating existing feature.

Is there a formed committee known as Change Control Board within the project? What user roles does it include? And if there isn't any CCB, who does approve major changes in the development of the project?

No, the project manager is responsible for approving changes.

Is there any formal form for change request?

no

Is there any Version Management tool deployed within any of the organization's projects' workspaces? If there is please name it.

SVN—VSS-- TFS

Is there any Software Configuration Management tool deployed within any of the organization's projects' workspaces? If there is please name it.

TFS

How do you describe the main roles and functionalities of the Version Management tool in the organization's project?

Versioning basics, we put source code into the TFS database and it handles the process of accessing them

How do you describe the main roles and functionalities of the Software Configuration Management tool in the organization's project?

Versioning, release management, visual studios is responsible for the build process

Is there any formal flow of Change Requests defined by the organization? If yes, please explain how the process is implemented.

The changes are directed to the team leader for assessment, and if the change is major, the request is redirected to the project manager.

What kind of users use the SCM tool if any exists?

All of the team members have access to the TFS tool to retrieve source code files

In the case of concurrent development, how are update conflicts being resolved?

We use exclusive locks, so the team cannot checkout the same item at the same time

Does the SCM tool handle the build and release process in the projects? If no, please specify what tool does the processes?

No visual studio handles the builds of the system

On what kind of circumstances does the organization branch off the main codeline in the project?

Major change will be taken into new branch for testing, new customer for an existing system

Would the baseline differencing and locking techniques proposed in our model be more useful than the traditional approach?

We work on projects that require agile development, hence it would be a useful addition to TFS and being able to lock the baseline and hide it from the view of the developers is a great addition for archiving old versions. Also it would give a more thorough managerial view of the change process itself.

Would the file version management proposed in our model be more useful than the traditional approach?

We use exclusive locks, so differencing would not be an issue since we do not have conflicts.

Would the change lifecycle proposed in our model be more useful than the traditional approach?

It would be helpful to implement this lifecycle in a tool, but for a company of our size, it would enforce unneeded overhead taking into consideration our development team size.

Interview B:

Person Name: Mahmood Alawneh

Person Position: Professional Services Manager

Company Name: MenaiTech

How do you describe the projects size developed by your organization?

Small to medium size, but mainly we develop one product in the company

How many developers are there in the company?

20

How many developers work on a typical project (average)?

20

What user roles are available in the project hierarchy?

Project manager, team leader, senior developer

How does the project team handle the process of change?

Most of the changes in our project are of the bugs kind, we receive a notification from the QA department and act upon it. Mainly the project manager is responsible for evaluating and approving the change; we use a bug-tracking tool to facilitate the process.

Is there a formed committee known as Change Control Board within the project? What user roles does it include? And if there isn't any CCB, who does approve major changes in the development of the project?

To be assembled this year. It will include a member from each major department and will be responsible for evaluating and approving changes across the project. We are highly interested in implementing a tool to automate the process.

Is there any formal form for change request?

A simple form to be filled by the QA department when a customer suggests a modification to our packaged product. The QA department handles the assessment and provides their feedback of the change.

Is there any Version Management tool deployed within any of the organization's projects' workspaces? If there is please name it.

TFS

Is there any Software Configuration Management tool deployed within any of the organization's projects' workspaces? If there is please name it.

TFS

How do you describe the main roles and functionalities of the Version Management tool in the organization's project?

Versioning management basics, storing source code and managing its versions.

How do you describe the main roles and functionalities of the Software Configuration Management tool in the organization's project?

Versioning management, and release management is handled by another software tool.

Is there any formal flow of Change Requests defined by the organization? If yes, please explain how the process is implemented.

Yes, but only for the changes that comes from the QA dept. which will go to the project manager to review, and if approved, the request is passed to the team leader to

What kind of users use the SCM tool if any exists?

The developers have access to the packaged product files through TFS.

In the case of concurrent development, how are update conflicts being resolved?

We use exclusive locks, and the team leader organize the work distribution between the developers

Does the SCM tool handle the build and release process in the projects? If no, please specify what tool does the processes?

The software releases are handled by a dedicated tool.

On what kind of circumstances does the organization branch off the main codeline in the project?

No branching for our project, we use a single-version strategy for all customers.

Would the baseline differencing and locking techniques proposed in our model be more useful than the traditional approach?

Yes it would be of much use since we depend heavily on virtual baselines during our product enhancements. Regarding the baseline locking it would be very useful since we have to deal with previous released versions by isolating them from the active versions.

Would the file version management proposed in our model be more useful than the traditional approach?

We do use exclusive locks on source code files and artifacts, so we do not utilize the file differencing.

Would the change lifecycle proposed in our model be more useful than the traditional approach?

We use a totally different approach when comes to changes, but being able to differentiate between problems and changes must come in handy, since we have lots of bugs fixing, and we don't want it to be mixed with added functionalities.

Interview C:

Person Name: Rami Arafat

Person Position: Technical Team Leader

Company Name: Esense

How do you describe the projects size developed by your organization?

Medium to large projects are presented here in our organization.

How many developers are there in the company?

Approximately 100 developers.

How many developers work on a typical project (average)?

50 developers.

What user roles are available in the project hierarchy?

Software developer, Team leader, Project manager, Technical manager and project director.

How does the project team handle the process of change?

We handle it verbally most of the time, and for major changes, we use paper work or project meetings.

Is there a formed committee known as Change Control Board within the project? What user roles does it include? And if there isn't any CCB, who does approve major changes in the development of the project?

We are not familiar with such terminology, most of the changes are reviewed and approved by the team leader, or in some cases the project manager or the technical manager. The approval depends on the level of change that is in hand, as the change gets bigger; the responsible user role is higher in the project hierarchy.

Is there any formal form for change request?

Yes there is. But the form fields are kept at minimal. It does provide the change requester name, and the affected files in the project, and the change description. It is mainly used internally between the team members.

Is there any Version Management tool deployed within any of the organization's projects' workspaces? If there is please name it.

Yes there is. We use Microsoft's Team Foundation Server (TFS) to handle version management.

Is there any Software Configuration Management tool deployed within any of the organization's projects' workspaces? If there is please name it.

Yes there is. We use Microsoft's Team Foundation Server (TFS) to handle version management.

How do you describe the main roles and functionalities of the Version Management tool in the organization's project?

Basically we use the TFS to store each version and revision of each source code file and other project-related documents. Basic version management features are also used, like branching but rarely.

How do you describe the main roles and functionalities of the Software Configuration Management tool in the organization's project?

We use TFS as a storage medium for project-related files and documents. Also we depend heavily on its Check-out paradigm (in most cases exclusive locks) to deal with concurrent development.

Is there any formal flow of Change Requests defined by the organization? If yes, please explain how the process is implemented.

The flow of change depends on the level of change, once a change becomes on the table, the team leader decides whether he should raise the change to an upper level, or to approve it himself. We depend on a flexible system when it comes to changes, not something automated, but acceptably efficient.

What kind of users use the SCM tool if any exists?

All users involved in the project use the TFS tool. Since all source code files are there in its main storage, all users must login to checkout files from the repository.

In the case of concurrent development, how are update conflicts being resolved?

We use exclusive locks, so no conflicts would arise.

Does the SCM tool handle the build and release process in the projects? If no, please specify what tool does the processes?

We use external program to deal with build and release management.

On what kind of circumstances does the organization branch off the main codeline in the project?

In the case of which a new customer with new requirements of an already-built software is introduced. We simply take a copy of the main development branch and a new branch is created with the new customer's requirements in mind.

Would the baseline differencing and locking techniques proposed in our model be more useful than the traditional approach?

We use the baselines concept heavily on our projects, and hence it would be interesting to see baselines differences related to change requests. But regarding the baseline locking I do not find it as useful as the baseline differencing.

Would the file version management proposed in our model be more useful than the traditional approach?

Of course, for developers it would be great help. But I'm not sure how it will be developed in real world.

Would the change lifecycle proposed in our model be more useful than the traditional approach?

It definitely adds more constraint to the change cycle itself, and I personally find it handful, but since we do not implement SCM in an automated way, the lifecycle won't be useful to us.

Interview D:

Person Name: Usama Abu Deyah

Person Position: Technical Team Leader

Company Name: Ejada

How do you describe the projects size developed by your organization?

Large (telecommunication based)

How many developers are there in the company?

800

How many developers work on a typical project (average)?

400

What user roles are available in the project hierarchy?

6-14 (Depends on the project nature and size)

How does the project team handle the process of change?

Changes in our projects are commonly major and created by a customer request. We use an internal process to gain proper evaluation and approval of the change in hand.

Is there a formed committee known as Change Control Board within the project? What user roles does it include? And if there isn't any CCB, who does approve major changes in the development of the project?

Yes. It contains the technical, business and marketing departments. Each change should pass through those departments to gain the approval.

Is there any formal form for change request?

Yes. Provided by IBM ClearCase

Is there any Version Management tool deployed within any of the organization's projects' workspaces? If there is please name it.

Yes , Subversion (svn)

Is there any Software Configuration Management tool deployed within any of the organization's projects' workspaces? If there is please name it.

Yes, IBM ClearCase.

How do you describe the main roles and functionalities of the Version Management tool in the organization's project?

- Manage the issue of multiple people contending for changing the same file at the same time.
- Keep sets of files each at a particular version. A version in the large of a collection of file versions. Attaching a label to show a baseline.
- Pull out a set of files for release that are all part of one specific labeled set or baseline.
- Save multiple baselines at different release versions in the repository simultaneously.
- Easily add a file item to the source control repository so that it can be revision managed and versioned controlled.

How do you describe the main roles and functionalities of the Software Configuration Management tool in the organization's project?

- Identify and document the functional and physical characteristics of configuration items.
- Control changes to configuration items and their related documentation.
- Record and report information needed to manage configuration items effectively, including the status of proposed changes and implementation status of approved changes.
- Audit configuration items to verify conformance to specifications, drawings, interface control documents, and other contractual requirements

Is there any formal flow of Change Requests defined by the organization? If yes, please explain how the process is implemented.

Yes,

- Business department starts a ticket upon the customer's request, or sometimes the ticket is initiated by internal business needs change.
- Ticket is handled by BRM (business request management) with a proper business impact analysis
- Ticket is redirected to the proper development team
- Technical Impact analysis is performed.
- The change is then implemented.

What kind of users use the SCM tool if any exists?

The team members have access to the project repository where project-related configuration items are stored.

In the case of concurrent development, how are update conflicts being resolved?

Won't happen because of the usage of the locking mechanism (exclusive locks)

Does the SCM tool handle the build and release process in the projects? If no, please specify what tool does the processes?

No, external software tool handles the build management

On what kind of circumstances does the organization branch off the main codeline in the project?

It happens when phasing out in house developed software with a new customizable product to satisfy new set of requirements.

Would the baseline differencing and locking techniques proposed in our model be more useful than the traditional approach?

A useful approach to deal with baselines and would add more managerial perspective to the concept of baselines.

Would the file version management proposed in our model be more useful than the traditional approach?

Yes it would helpful, but we use exclusive locks, so it will not affect our environment.

Would the change lifecycle proposed in our model be more useful than the traditional approach?

It segregates the change lifecycle to include more than one type of change. And gives the flexibility to a company to apply constraints for change control as fits its business best.

Interview E:

Person Name: Samer Qaisi

Person Position: Quality Assurance Manager

Company Name: ATS

How do you describe the projects size developed by your organization?

Medium to large

How many developers are there in the company?

50 team members

How many developers work on a typical project (average)?

15

What user roles are available in the project hierarchy?

Project manager, business analyst, implementers, team leader, senior developer, junior developer

How does the project team handle the process of change?

Team leader will evaluate the change; if the change was business related he will escalate to the project manager and business analyst to evaluate. If not, he will handle the change internally with his team members.

Is there a formed committee known as Change Control Board within the project? What user roles does it include? And if there isn't any CCB, who does approve major changes in the development of the project?

Yes, project manager, team leader and business analyst

Is there any formal form for change request?

Yes. Used internally to satisfy new customer needs.

Is there any Version Management tool deployed within any of the organization's projects' workspaces? If there is please name it.

TFS - VSS

Is there any Software Configuration Management tool deployed within any of the organization's projects' workspaces? If there is please name it.

TFS

How do you describe the main roles and functionalities of the Version Management tool in the organization's project?

Versioning basics and store source code files in a dedicated project repository

How do you describe the main roles and functionalities of the Software Configuration Management tool in the organization's project?

Versioning, storage, tasks management, document management

Is there any formal flow of Change Requests defined by the organization? If yes, please explain how the process is implemented.

The changes are directed to the team leader for assessment, and if the change is major, the request is redirected to the project manager.

What kind of users use the SCM tool if any exists?

All team members have access to TFS

In the case of concurrent development, how are update conflicts being resolved?

Manual conflict resolution handled by the team leader

Does the SCM tool handle the build and release process in the projects? If no, please specify what tool does the processes?

No. the project manager handles the system's full builds using specialized tools

On what kind of circumstances does the organization branch off the main codeline in the project?

We do not branch. We copy the files to a new project repository and initiate a new project each time a new customer requests a product.

Would the baseline differencing and locking techniques proposed in our model be more useful than the traditional approach?

We do not use baselines in our projects frequently. But it would be a great addition.

Would the file visual differencing proposed in our model be more useful than the traditional approach?

It is useful especially that we use manual conflict resolution. It would be a great addition.

Would the change lifecycle proposed in our model be more useful than the traditional approach?

It would be useful to make multiple approaches according to the change type. But maybe having only two parts, PR and CR would make more sense.

Interview F:

Person Name: Bilal Audeh

Person Position: Team Leader

Company Name: TechnoSIS

How do you describe the projects size developed by your organization?

Medium-sized business (insurance management packages)

How many developers are there in the company?

20

How many developers work on a typical project (average)?

10

What user roles are available in the project hierarchy?

Project Management, Project Coordinator, Project implementer, Project Consultant, Developer

How does the project team handle the process of change?

Through a Gap Analysis Document (Client's IT team prepares it after project plan is fully implemented and after system trial run and before live run)

Is there a formed committee known as Change Control Board within the project? What user roles does it include? And if there isn't any CCB, who does approve major changes in the development of the project?

No there isn't, Project manager from our side and project coordinator from the client side does the needed approvals for changes

Is there any formal form for change request?

Yes

Is there any Version Management tool deployed within any of the organization's projects' workspaces? If there is please name it.

Yes, VSS

Is there any Software Configuration Management tool deployed within any of the organization's projects' workspaces? If there is please name it.

No

How do you describe the main roles and functionalities of the Version Management tool in the organization's project?

Keeping track of all versions at all times and who should/should not have it.

How do you describe the main roles and functionalities of the Software Configuration Management tool in the organization's project?

No SCM tool is implemented in our environment.

Is there any formal flow of Change Requests defined by the organization? If yes, please explain how the process is implemented.

no

What kind of users use the SCM tool if any exists?

Our developers have access to project database, and the project manager and team leader.

In the case of concurrent development, how are update conflicts being resolved?

Back to documentation (manual conflict resolution)

Does the SCM tool handle the build and release process in the projects? If no, please specify what tool does the processes?

No, we use a dedicated software for the build process.

On what kind of circumstances does the organization branch off the main codeline in the project?

We do not use the concept of branching.

Would the baseline differencing and locking techniques proposed in our model be more useful than the traditional approach?

Yes it would be helpful to see what changes were made between each baseline, and what was added of functionalities.

Would the file visual differencing proposed in our model be more useful than the traditional approach?

It is useful especially that we use manual conflict resolution.

Would the change lifecycle proposed in our model be more useful than the traditional approach?

The only source of changes is through our customers' due of change of business needs. The approach is not relevant to our development methodology.