CONFIGURATION MANAGEMENT FOR A DISTRIBUTED AND COLLABORATIVE SOFTWARE DEVELOPMENT ENVIRONMENT

BY

ENG

TERESA LIU

B.S., ENVIRONMENTAL ENGINEERING SCIENCE (1999)
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Submitted to the Department of Civil and Environmental Engineering in partial fulfillment of the requirements for the degree of

Master of Engineering in Civil and Environmental Engineering

AT THE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2000
[June 2000]

© 2000 Teresa Liu. All rights reserved
The author hereby grants to MIT permission to reproduce and distributed publicly paper and electronic copies of this thesis document in whole or in part.

AUTHOR... [illegible]
DEPARTMENT OF CIVIL AND ENVIRONMENTAL ENGINEERING
MAY 18, 2000

CERTIFIED BY.. [illegible]
FENIOSKY PEÑA-MORA
ASSOCIATE PROFESSOR OF CIVIL AND ENVIRONMENTAL ENGINEERING
THESIS SUPERVISOR

ACCEPTED BY... [illegible]
DANIELE VENEZIANO
CHAIRMAN, DEPARTMENTAL COMMITTEE IN GRADUATE STUDIES

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
MAY 3 0 2000
LIBRARIES

ENG

CONFIGURATION MANAGEMENT FOR A DISTRIBUTED AND COLLABORATIVE SOFTWARE ENGINEERING ENVIRONMENT

By

TERESA M. LIU

Submitted to the Department of Civil and Environmental Engineering on May 18, 2000 in partial fulfillment of the requirements for the degree of Master of Engineering in Civil and Environmental Engineering

Abstract

In the face of changing technology, the world is becoming more global by the minute. This globalization has resulted in dispersed teams and collaborative opportunities, which offer value and depth to projects. Software development also reflects this globalization, as do all the processes involved in software development. It is important to understand these changes and how they affect not only the entire development process as a whole but also each individual part of the process.

Configuration management is an integral part of the software development process. In order to perform good software development, it is imperative to understand and be able to implement proper configuration management. Thus, as configuration management is important to software development, it is also important to study the effects of how the changing context of software development toward distributed and collaborative environments affects configuration management. Not only does configuration management influence the transition to this type of environment, it also is quite affected by it.

In this thesis, I aim to examine the impact of a distributed and collaborative development environment on configuration management. I will first give an overview of software engineering with respect to configuration management; next, I will provide and introduction to traditional configuration management, and in the following chapter, I will discuss distributed and collaborative configuration management as it exists today. Next, I will provide a case study of the ieCollab project, a project in which developers were dispersed and a great deal of collaboration occurred, and finally, I will discuss the future of collaborative and distributed configuration management.

Thesis Supervisor:	Feniosky Pena-Mora
Title:	Associate Professor of Civil and Environmental Engineering

Acknowledgements

I would like to first thank my parents Ed and Caroline Liu for their love and support and for always providing me with the best, and my brother David for relieving my stress.

I would also like to thank Professor Feniosky Pena-Mora for his guidance on this thesis.

Great thanks also go out to my spiritual family at ABSK and Berkland Baptist Church. I couldn't have made it through five years at MIT without your love and prayers.

Not to be forgotten are the MEng folks – thank for the memories…

Finally, I thank my Lord and Savior Jesus Christ for His faithfulness. To Him be the glory.

This thesis is dedicated to the memories of my aunts Liu Lin and Chang Meng-Li.

Table of Contents

List of Figures

List of Tables

1. Introduction

With the advent of the twenty-first century, we find the world becoming a smaller and smaller place. The globalization of businesses is occurring at an astonishing rate, information is more readily available than ever before, and suddenly fellow engineers, colleagues, and even students on the other side of the world are only a mouse-click away. Information Technology has been a mighty hand in these developments. The commoditization of the personal computer, faster connections, and growing affordability of technology in general are really making this the "information age."

As a result of the changing technology, the ways in which people live and work are being redefined. For example, e-mail, which allows asynchronous yet immediate transfer of electronic mail and enables people to communicate more efficiently, and the Internet and the World Wide Web literally provide information at our fingertips. Physical distance also no longer hinders us from working and communicating with others halfway across the world, and the sharing of resources that is now possible enriches the experience of each person involved. It is safe to say that nearly every industry has been or can be affected by IT.

1.1 Motivation

The software industry is an industry that IT has affected in several different aspects. Not only is the software itself IT, but the process in which the software is developed is very dependent on IT. Just like in any business, technology now allows software developers to be geographically dispersed and yet work closely together, and this situation is allowing software projects to grow in both complexity and size. However, even as this environment of distributed location and considerable collaboration have become more popular and even commonplace, many projects still fail or experience massive delays. These problems occur for a number of reasons, including the following (Williamson, 1990):

- increase in chances for error as complexity and size of the project increase
- difficulty in synchronization as the number of possible communication paths among a group increases more for each new person.
- inefficiency and too much overhead as a result of dispersed teams

Thus the question is raised-- in such an environment, how can such software development projects best be accomplished? It is obvious that the entire software development process must change in order to remain effective in a distributed and collaborative environment. How can these processes be accomplished the most effectively? Although there is no one solution to these problems, there is a set of procedures that can help in alleviating them. This set of procedures is referred to as configuration management, and it is the motivation behind this thesis. A key process in software development, configuration management will be particularly important in facilitating the transition to this environment. Not only will it be very influential in this change, but configuration management itself will also be greatly influenced by it.

1.2 Overview

This thesis will be structured in the following manner. First, a background on software engineering, software development, and changes in the software development environment will be provided in Chapter 2. A description of the configuration management process, its role in software development, and the activities involved will then be given in Chapter 3. The next topic of discussion, in Chapter 4, will be how configuration management is affected by the move toward distributed and collaborative environments, and following this discussion, in Chapter 5, I will present a case study on ieCollab. ieCollab is an MIT-based software development project designed to study this topic of distributed and collaborative environments. I would also like to give suggestions for the improvement to configuration management in the context of the ieCollab project. The last portion of the thesis, Chapter 6, will be devoted to the future of configuration management in the face of changing technology. I will try to provide answers to questions such as what kind of methodology should be used, as well as what kinds of tools would be useful to configuration management.

2. Software Engineering

Configuration management is a key step in software development. Therefore, to truly understand configuration management, it is first important to understand the software engineering context in which it is carried out.

Over the past forty years, software engineering has gone through many changes. At present, desktop computers are quickly reaching the state where they have become commodities, and it is predicted that computers will soon be in even common household items (D'Amico, 1999). As the prevalence of computers grows, so will the demand for software. The field of software engineering is one of the fastest growing fields, both in profit and development. Whereas software engineering was an uncertain and rather frail area at its inception, today it is recognized as a legitimate discipline and a process that merits serious research and study (Pressman, 1997).

The software engineering discipline is comprised of a set of definitions, processes, and frameworks, all of which will be described in this chapter. For example, the Capability Maturity Model is a well-known organizational framework for software development processes. These processes, which are fairly standard across all development projects, can be modeled differently according to the project needs and objectives and can occur in different development environments as well.

2.1 The Capability Maturity Model

As stated earlier, Configuration Management is a very important process in software engineering. Not only is it a key process in software development process, it is also a key process in the Capability Maturity Model.

2.1.1 Overview of CMM

One problem identified by the careful study of software engineering over recent years is the difficulty of managing the software development process. A software process is defined as "a set of activities methods, practices, and transformations that people use to develop and maintain software and the associated products" (Paulk, et al. 1997). This definition has been put into practice by the Software Engineering Institute (SEI), which has developed a framework for processes employed by software organizations for developing and maintaining software. This framework is called the Capability Maturity Model (CMM). The basic principle behind the CMM is that an organization's software process becomes better defined and more consistently implemented as the organization matures. By helping the organization determine its current process maturity and also identifying the most critical areas for improvement, CMM is able to provide a model for continuous and evolutionary improvement on the key process areas in software development (Paulk, et al. 1993).

According to the CMM, the maturity of software processes can best be achieved in incremental and evolutionary steps. An immature software organization's processes are reactionary and inconsistent, and the product unpredictable. However, the processes of a mature software organization are well defined and followed throughout the entire organization, and the quality of the product is closely monitored. However, in order to gage what kind of improvements it ought to make, an organization must be able to determine its current maturity level.

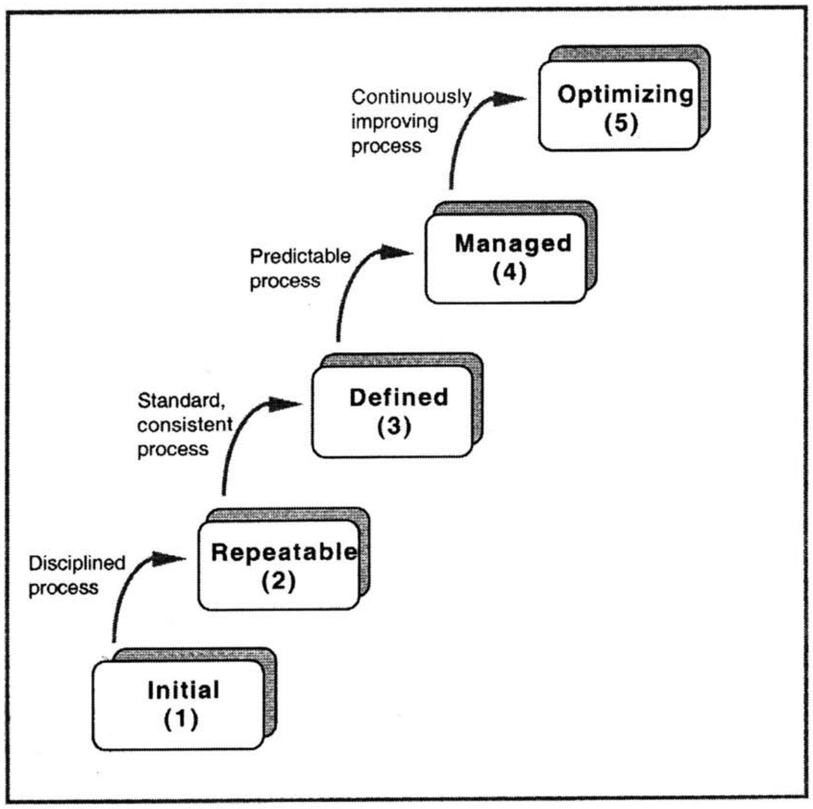


Figure 2-1 Five Levels of Process Maturity, Capability Maturity Model (Paulk et al, 1993)

The CMM defines five levels of maturity for software processes, and each level is characterized by key process changes that should be made to improve the process. These levels are illustrated in Figure 2-1. Level 1 is the Initial Level, the level of process maturity at which there are few stable processes and performance is not dependent on the organization but rather the talent of the individuals. Level 2 is the Repeatable Level. At this level, the organization has implemented basic but stable processes for project planning and tracking and can repeat its previous successes. Thus its software process is disciplined and its management is effective and operates based on past successes. The next level defined by the CMM is Level 3, the Defined Level. At this level, the entire organization follows a documented standard process for developing and maintaining software. Thus both management and software engineering practices are well defined, stable and repeatable. The fourth level of maturity in the CMM is the Managed Level. At this level, the software process capability of the organization can be considered predictable as a result of the measures taken to set goals and make quantitative evaluations. In addition, the product will be of predictably high quality. The last level of software process maturity is Level 5, the Optimizing Level. This level is characterized by continuous improvement, with the whole organization committed and acting proactively to improve the development process. This improvement occurs as a result of learning from defects and also new innovations (Paulk, et al. 1993).

2.1.2 Role of Configuration Management in the CMM

The CM process is positioned differently in each level of the CMM. In an organization at the Level 1 stage, a CM process has not yet been established or has just been established. Thus, this organization has no successful CM experiences to look back on and imitate. If CM is implemented, success depends on the talent and insight of those who implement it.

Along with requirements management, project planning, project tracking, subcontract management, and quality assurance, configuration management is a key process area for Level 2 in the Capability Maturity Model. CM practices contribute to the basic level of discipline characteristic of Level 2 process maturity. This discipline is supported by the goal of CM, maintaining the integrity of the product during the product evolution. CM also helps to establish basic project management controls, offering the project the stability necessary to repeat earlier successes and thus enable an organization to reach the "Repeatable level" in process maturity (Paulk et al. 1993). At Level 2 maturity, for example, the organization may have identified key items to be controlled and a simple CM project repository and protocol for retrieving and replacing these items based on previous experience, and change may be handled in a similar fashion.

Once an organization has reached Level 3 process maturity, CM processes have been standardized and made an integral part of the development process (Paulk, 1993). For instance, the organization will most likely have employed the IEEE standards (IEEE 1990) to create the organization-wide CM standards. Furthermore, at Level 3 maturity, an organization most likely has designated a team of developers to oversee CM and manage the administration of the project repository. A structured change control process will also have been standardized, along with the instantiation of a Change Control Board for approving changes.

At Level 4, the Managed Level, the organization examines the quality of the resulting product as a way of measuring CM success and efficiency. The CM process is both understood and controlled, and the organization may, for example, utilize an automated tool to aid in making the process more efficient and orderly. Finally, at Level 5, the Optimizing Level, the organization is looking into ways to optimize the CM process, investigating innovative technologies for CM and disseminating knowledge on lessons learned.

2.2 The Software Development Process

The software development process is a very important aspect of software engineering, and it consists of several activities. These activities apply to all software projects, regardless of the project complexity or size. These activities can occur at different stages during the project and have different purposes, according to the individual project.

2.2.1 Software Process Activities

The process activities for a software development project can be organized into two types of activities-- framework activities and umbrella activities. These framework and umbrella activities make up the common process framework. Each framework activity is made up of task sets; these tasks sets include all work tasks, milestones and deliverables, and software quality assurance points (Pressman, 1997). Figure 2-2 illustrates the common process framework.

2.2.1.1 Framework Activities

Framework activities are activities that are integral to the production of the software and include the following activities:

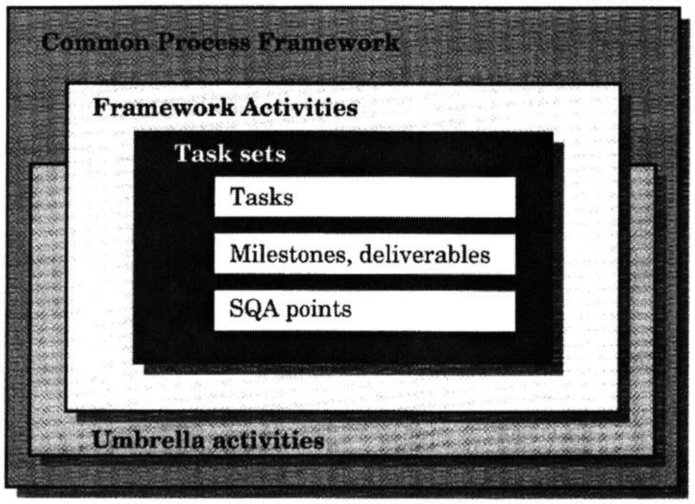


Figure 2-2 Common Process Framework (Pressman, 1997)

System engineering

System engineering is the process that must occur before any plans for the software begin. It uses a collection of top-down and bottom-up methods to examine not just the software, but the entire system and the elements that will work together to help achieve the goal of the product. System engineering activities include communicating with the customer to understand the customer's desires and needs for product, evaluating the feasibility of these needs, and creating a system definition that provides the foundation for subsequent engineering work (Pressman, 1997). In some organizations, a "business manager" or "marketing manager" may handle these activities.

Requirements Analysis

According to Pressman, requirements analysis is a software engineering task that bridges the gap between the system engineering process and software design (Pressman, 1997). Working with system engineers and based on previously defined needs, the primary task of requirements analysis is to specify software function and performance, the interface with other parts of the system, and constraints. Requirements analysis should then provide models to software designers that can be translated into design.

Software design

Simply stated, the software design process essentially takes the "what" provided by requirement analysis and describes "how" it should be done. Pressman (1997) defines software design as "an iterative process through which requirements are translated into a 'blueprint' for constructing the software." It refines the requirements already specified and provides the detail necessary for implementation.

Component Implementation

After the design process and modeling is complete, coding on the actual product can begin. The design documents should provide enough specificity to produce algorithms, and the requirements analysis specifications provide a reference for quality control. This step in the software development process is arguably the one of the most important, as it is the step where the actual product is produced (Scacchi, 1987).

Software Testing

Software testing occurs after the software product has been built. The main focus of testing is to discover errors in the program, errors that will undoubtedly be present. Although the testing concept is quite simple, good testing requires careful planning and consideration of the requirements and should be planned before coding even begins.

2.2.1.2 Umbrella Activities

Umbrella activities can be described as activities that occur throughout the software development process and not just at a particular time. Thus, they occur independently from the framework activities described above. Umbrella activities include project management, quality assurance, and configuration management.

Project management

As with any project, project management for a software project is a very important activity. Project management activities manage three areas of a project: people, problem, and process. The project management must ensure that there are appropriate people for performing project tasks, continually ensure that the project objectives are being met, and that the process being used to achieve these objectives is correct. Project management is also responsible for defining a project plan and scope and scheduling (Pressman, 1997).

Quality Assurance

The simple explanation for the purpose behind quality assurance is the production of high-quality software. This goal can be achieved through proper quality assurance by numerous activities applied throughout the project duration, such as technical reviews, testing, monitoring the proper adherence to standards, and measurement and reporting.

Software Configuration Management

Like the other umbrella activities, configuration management is applied throughout the software development process. The main concern of configuration management is *change* in the development process. These changes could include changes to code, specifications, data, and other items. Thus the activities of configuration management include identification, control, management, and reporting of changes. Because CM controls all the items of a project, it is often considered the "memory" of the project. However, for some organizations, a specially designated group handles the collection of project knowledge. This group will be referred to as the Knowledge Management team in this thesis. The role of configuration management in the software development process will be discussed in the following section of this chapter.

2.2.2 The Role of CM in the Software Development Process

The main responsibility of CM in the Software Development Process is to ensure the controlled evolution of the software product and to make sure that this evolution is carried out in a manner which will preserve the product's integrity. The role that CM takes in the software development process is a supporting role. It is often viewed as a process closely related to project management, as well as quality assurance. However, CM is important in its own right. It does not produce any part of the product, but instead helps to manage risk. This is done in the CM process this by using technology and management to:

- Identify the important system components discrete points in time and store these components
- Record and trace changes to the system components
- Provide tools for controlling these changes
- Verify these changes and the way in which they were made

Along with managing risk, CM activities also facilitate reuse of system components and repeatability of the project.

Thus, within the software development process, CM is an umbrella activity, not associated with any particular framework activity but affecting all aspects of the software development process. It is not performed at any particular time but is instead an ongoing process. For example, in some of the software process models described in Chapter 2 such as the waterfall method, many of the software development processes occur only at certain points in time, such as after the previous process has been completed. However, CM occurs at all times and is not dependent on any other processes.

In addition to integrity, CM also provides measures for accountability, visibility, reproducibility, coordination, and traceability (Ben-Menachem 1994).

2.2.3 Software Process Models

Every software project must adopt a development strategy depending on the nature of the project and product, as well as tools and methods being used. To determine the right strategy for a project, it is important to first look at the different activities involved in software development and how they relate to one another and the overall process. Process models provide a way to do this by organizing the activities into different stages and defining how and when they should be done. Thus, these process models attempt to bring some structure and order to the otherwise very chaotic software development process and organize the software process into task chains. These task chains are "idealized plans of what actions should be accomplished and in what order" (Scacchi, 1987). There are many process models, each offering a different view on the software process, and configuration management, being an integral part of the development process, has part in each of these models. A few examples of process models are discussed below.

2.2.3.1 Linear Sequential Model

Also known as the "waterfall model" or "classic model," the linear sequential model emphasizes a sequential approach to the stages in the development process. Modeled after the conventional engineering cycle, the key activities in this model are analysis, design, coding, testing, and maintenance, and they occur in this order. According to the linear sequential model, the next activity in the process does not begin until the previous one has been completed (Scacchi, 1987). Figure 2-3 illustrates the flow from one activity to the next.

Configuration management activities occur during the entire life of the project; however, they occur mainly at the end of stages, typically when items under CM control become finalized. For example, after the analysis stage has been completed, CM will place approved items in the project repository. However, once the design phase is complete and once coding and testing begin, CM activities may occur continually, as change occurs frequently (Bersoff and Davis, 1991).

Although it is the oldest and most widely used model, the linear sequential model does have its disadvantages. For example, there is often a great deal of uncertainty at the beginning of a project, making it difficult for the first steps to be completed quickly or accurately. This inefficiency will affect the rest of the process, since each consecutive step must occur only after the previous one has been completed. Furthermore, this model also does not make the best use of resources, as teams working in a later stage of the cycle can not work until the previous stages have been finished. Along the same lines,

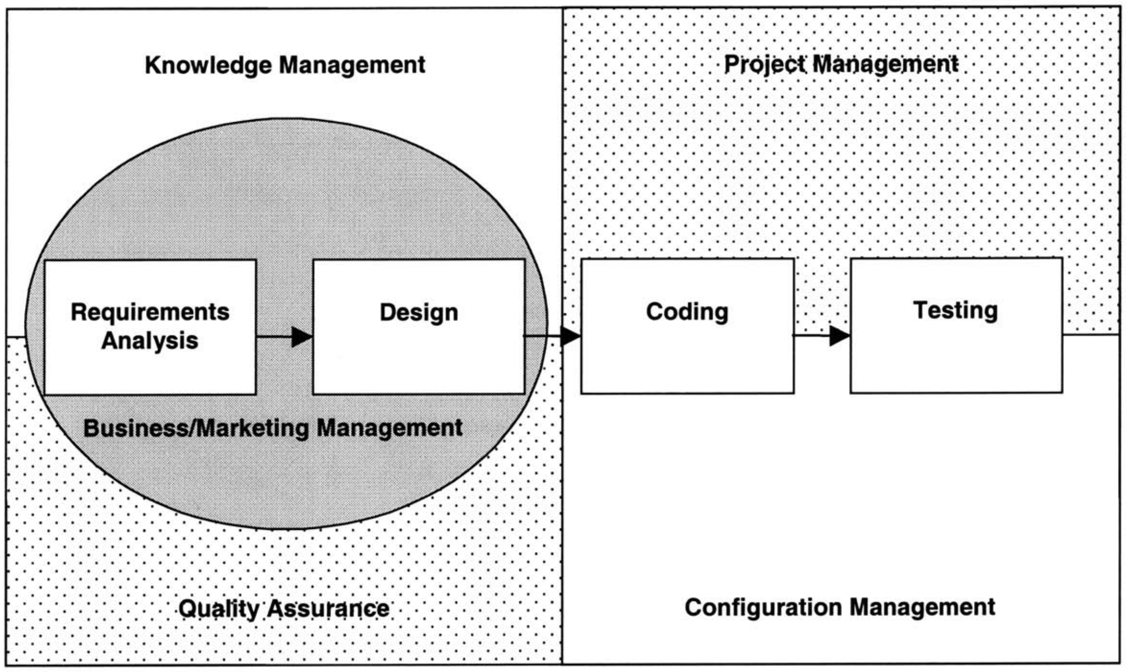


Figure 2-3 Linear Sequential Process Model, (modified from Pressman, 1997)

the pure linear sequential model also does not allow for iteration, which is often necessary when mistakes are made and changes are needed. Because of the successive nature of the process, developers also often experience many unnecessary delays. Another disadvantage of the linear sequential model is the unavailability of a working model of the product until the end of the project life span (Pressman, 1997). Finally, the linear sequential model may not be appropriate for distributed and collaborative development, since using this model is very deliberate and often time-consuming for the above-mentioned reasons, and the project speed would not be helped by the additional time that is required for collaborating in a distributed manner.

Although this process model does have a number of disadvantages, it also has many advantages and, thus, is still widely used. One advantage of the linear sequential model is it's providing of an orderly sequence of transitions from one stage to the next in a linear fashion. This order provides a much-needed structure to the software process, which allows projects to be rather straightforwardly managed, which is especially beneficial in large project situations or even distributed situations. This model is also very easy to implement (Pressman, 1997). Furthermore, because of the sequential nature of this model, CM resources are not strained, allowing other energy for other goals, such as collaboration.

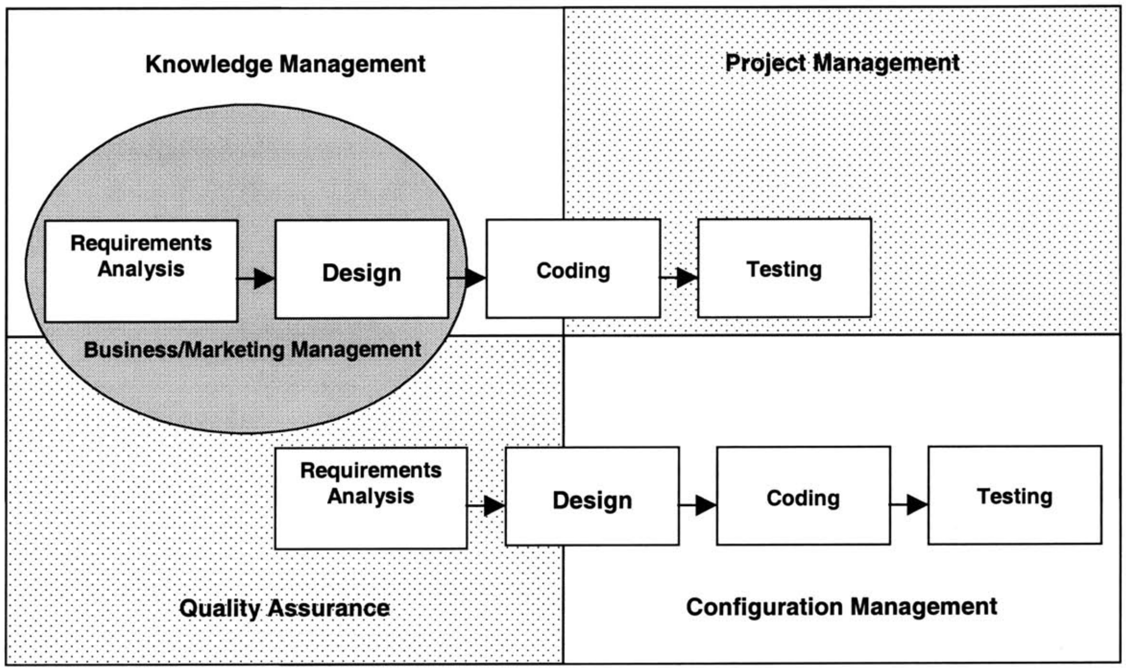


Figure 2-4 Incremental Process Model (Modified from Pressman, 1997)

2.2.3.2 The Incremental Model

The incremental model applies the sequential philosophy behind the waterfall method while introducing iterative and prototyping concepts. As shown in Figure 2-4, the incremental model employs staggered linear sequences. The main concept behind this model is the delivery of the product in increments. A project using the incremental model may deliver the product in modules, where each module provides a different functionality in the product. However, the first delivery typically has the essential operating functions, and subsequent deliveries add more features (Scacchi, 1987). After delivery of each portion, the customer may evaluate the module, and provide suggestions for what to improve for the next iteration. The process is repeated until the final product is delivered.

Configuration management activities occur much more frequently throughout the process life cycle in the incremental model than in the linear sequential model. Although CM activities are concentrated at the end of each stage and during the actual production of the product as they are in the waterfall model, CM, because the incremental model is iterative, there are more opportunities for change, which is a key motivation for CM. To elaborate further, according the linear sequential model, changes only occur when a problem is found or an enhancement needs to be made. However, according to the incremental model, not only are more iterations of the stages performed, giving opportunities for new configuration items, but there are also more opportunities for changes and modifications.

One of the main advantages of the incremental model is that it allows for feedback from clients. This feedback can be extended so that other members of the development team provide the feedback as well. This structured feedback is especially useful in the case of collaborative development. Because the structure for feedback is already in place in the process model, less initiative is needed to generate feedback on the product. The incremental model also allows for better utilization of people and resources than the linear sequential model does since different iterations of the cycle are being performed in parallel yet staggered fashion (Pressman, 1997). This is particularly useful for a distributed development environment because it provides more opportunities for collaboration between stages.

However, though the incremental model does allow for better use of resources, this allocation of resources may be difficult to coordinate, especially in a distributed environment when resources are located in dispersed locations. This situation would require a great deal of coordination and effort. Furthermore, the iterative nature may cause a straining of CM resources at times, which could lead to disorganization and confusion in the project. However, even with the disadvantages, the incremental model may be one of the best choices for life cycle models, and the large number of commercial firms using this model for software development evidences this (Scacchi, 1987).

2.2.3.3 The Spiral Model

Like the incremental model, the spiral model for software development is an evolutionary process model that addresses many of the challenges raised by the traditional linear sequential model by applying the systematic and controlled aspects of linear sequential model while giving it an iterative and prototyping nature. The spiral model considers a set of six framework activities, or task regions, as listed below (Pressman, 1997):

- customer communication
- planning
- risk analysis
- engineering
- construction and release
- customer evaluation

Figure 2-5 illustrates the spiral model. The process begins at the center of the spiral and moves through each task region in the clockwise direction. After completing an iteration of the cycle, a new one will begin based on the feedback received from the customer. The inner cycles of the spiral represent more analysis and prototyping, where the product of the first circuit may result in a product specification. Subsequent circuits of the cycle follow the linear sequential model more closely, resulting in prototypes and more complete versions of the product. Another important feature of the spiral model is that it includes additional customer communication and evaluation steps at the end of each pass through the cycle. Using the spiral model, a software product is developed in iterations of the six framework activities, where the products developed in iterations begin as prototypes and get increasingly complicated and complete with successive iterations (Pressman, 1997).

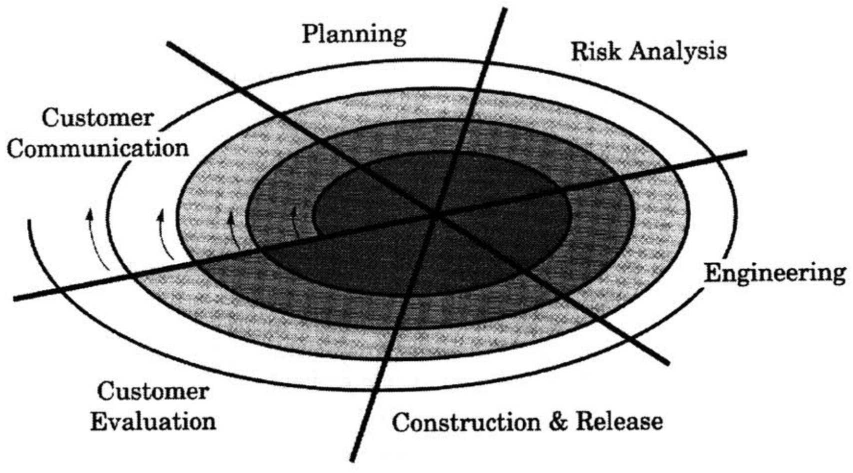


Figure 2-5 Spiral Process Model (Pressman, 1997)

In the spiral model, CM activities are heavier in the outer levels of the spiral than in the inner layers, since the outer cycles of the spiral take on the characteristics of the classic linear sequential life cycle and the inner layers produce less in terms of CM items. Furthermore, since what is produced at the end of each cycle is typically a prototype that is "thrown away" and not changed, CM activities may be heavy with respect to handling new configuration items. As with any iterative model, CM will also have to be very active during the entire life of the process, especially as customers provide feedback that will necessitate change.

The spiral model is a very realistic approach to developing large-scale systems and software (Pressman, 1997). The customer feedback and iterative approach addresses many of the challenges associated with the linear sequential model, such as wasted time spent waiting for one activity to be finished and the discovery of errors. This model of feedback is very useful for collaborative projects and takes risk into account as a large consideration, which is also useful for collaborative and distributed projects, which are often high risk. However, this process model takes a good deal of resources and management to implement, and it is also still a relatively new model. Therefore, although useful, it may be difficult to implement in a distributed environment.

2.2.3.4 Other Process Models

Other process models offer similar strategies for the software process, often adapting another model to specific needs. For example, the Rapid Application Development (RAD) model is an adaptation on the linear sequential model and is designed particularly for projects where a very short development cycle is necessary (Pressman, 1997). In this model, CM is especially important since all steps are being done at a very fast pace. In a very fast-paced environment, CM practices are often done in a "quick and dirty" method, which can result in complications later (Bersoff and Davis, 1991). The component assembly model is another process model, based on the spiral model. However, it tries to reuse software and components to reduce the cycle time by up to 70%. Reuse complicates CM in this model, however, especially if the reused module is used by more than one product. These two models, although producing results very quickly and efficiently, require a great deal of management and commitment from those involved (Pressman, 1997), qualities that are often harder to generate in distributed environments.

2.3 Software Development Environments

The software development environment is another important aspect of software engineering, and it encompasses many aspects of the software project. Understanding the development environment is crucial to every project. According to Brown et al. (1992), a software development environment is "a complete, unifying framework of services supporting most (or all) phases of software development and maintenance." This means that a description of a project's software development environment will consider the project's technical, managerial, and even political environments. The mechanisms, tools, and processes used to address these issues are also a part of the development environment.

In more recent years, since the popularity of computers and need for software has grown and there have been many changes in software engineering in general. Thus, software development environments have taken on new characteristics, as well as new needs.

2.3.1 Collaborative Software Development Environment

One type of environment that has very prevalent is a collaborative software development environment. As software projects have become increasingly complicated, there has been a greater need for collaboration in the software development process. The sheer size of projects has made it necessary to divide work between many developers, and organize developers by processes. Furthermore, the value of teamwork has also been particularly emphasized in recent years (Belanger et al, 1996). Thus, working jointly with other developers and teams has become quite standard. The main focus of this collaboration is sharing. This involves the sharing of ideas, documents, project modules that interact with other modules that need to be integrated form larger ones, code for testing, as well as other items, all within a team or between teams. Collaboration can even involve groups of people from completely different organizations or backgrounds working together for one purpose.

The benefits of collaboration include allowing for enriching and enhanced experiences that would not occur with individual work. Most people generally agree that better results are achieved in collaborative settings, with more ideas being generated and more feedback being given. The challenges of having a collaborative development environment are mostly made up of the difficulties that come with many people working together. For example, communication problems, differences of opinion, different working styles, and personnel management all impact the success of a collaborative effort.

As stated, collaboration is a very important part of software projects and is becoming easier with advances in technology. Furthermore, the success of the collaboration is often a very important factor for whether a project is successful and delivers on time.

2.3.2 Distributed Software Development Environment

Complementing the now common collaborative software engineering environment, distributed software development environments are also being quite readily adopted. "Distributed environment" can mean many things. One common use of this term at present refers to the notion of shared services provided transparently by cooperation between remote and local processors (Kramer, 1994). In this client-server environment, clients may make requests for service, which is allocated by a server and relinquished after use. Although "distributed environment" may sound as if users are far apart from each other, in most cases, users are in the same office, building, or local area. The benefits of this kind of distributed development environment include the benefits that come with shared services and centralized services, such as lower costs and fault tolerance (Kramer, 1994).

A "distributed software development environment" encompasses this concept of distributed systems but includes another level of distribution. This additional level of distribution refers to a group of people who are geographically distributed over a *considerable* distance and yet have some common objective. Although it is quite ordinary for two developers to be in offices across the street from each other and working on the same team, truly physically distributed teams are still not as widespread, though they are becoming more so. However, this kind of physically distributed development environment can only occur because of the technology of the distributed systems discussed earlier, which have made it feasible for two developers to be in different countries and work together. Although this allows for a greater amount of collaboration, integrating and coordinating distributed systems requires a great deal of effort. This effort involves many choices as to how to configure hardware and software to achieve successful distributed collaboration. In addition, such projects also introduce managerial challenges (Belanger et al, 1996).

2.3.3 Distributed and Collaborative Software Development

The kind of distributed effort just described can only be achieved when done hand in hand with collaboration. However, even though collaboration is necessary in a distributed environment, the amount of collaboration that actually occurs may vary from case to case. For example, one level of distributed collaboration occurs when a single organization has offices in many different and distributed locations. In a software development situation, this would probably mean communication and sharing of resources between different sites yet having local systems that are tailored to each location. However, another example of collaboration in a distributed environment would be a case where several entities from *different* organizations collaborate together on the same project. This phenomenon has been termed "virtual enterprises," and it is a situation that is just beginning to occur in the software development industry (Noll and Scacchi, 1997). Organizations in virtual enterprises usually require that some autonomy over their own processes and tools; however, they still need to adopt common practices to collaborate and work with other dispersed developers. These needs of distributed collaborative software development environments are the needs that ought to be considered when discussing software engineering for the future.

Thus, distributed and collaborative software environments are becoming more and more important, especially with the globalization of companies and increase in collaboration across the world, which necessitates tools for long distance collaboration. However, distributed and collaborative environments also introduce many challenges to software engineering, both technical and non-technical. Tackling these challenges is not an easy task; it is difficult more often than not and requires a great deal of planning and forethought.

2.3 Summary

In summary, software engineering is a very structured discipline that must be understood before understanding configuration management. Many tasks and activities must be accomplished by a software development project before a product is produced, and the collection of these activities is referred to as the process. Configuration management is only one of the processes in software development. Depending on the level of structure it has reached in its development process and how consistent and deep its activities are, an organization can be categorized into different levels of process maturity according to the Capability Maturity Model.

Each type of activities in the software development process serves a different purpose in a project. In addition, the processes that these activities are a part of occur at different stages within the life of a project, and although there is a basic order to these stages, the actual order depends on an individual project's goals, objectives, and environment. The order and interaction of these processes can be modeled in many ways. Furthermore, the environment in which they are performed also influences the processes and activities in a development project.

Finally, configuration management has a position in each aspect of software development, from the process to the life cycle. In order for one to truly understand configuration management, it is necessary to understand the concept of software development. The next chapter, Chapter 3, will explain the specific principles and practices that make up configuration management.

3. Software Configuration Management

Although it is often viewed as an overhead expense and a "non-productive" part of any software project, Software Configuration Management, hereafter known as Configuration Management, is an integral and vital part of every software project. This chapter will discuss the details of configuration management—what problem it addresses, an overview of how it is done, and the specific activities that help to accomplish its goals.

3.1 Purpose and Motivation of Configuration Management

The purpose of configuration management (CM) is to maintain the integrity of a product as it evolves to make its evolution more manageable (Buckley, 1993). As evolution implies change, the main source of difficulties that corrupt integrity in a software project's configuration management is change. However, change is inevitable in any project, and thus, maintaining a protocol for change and evolution is very important to the success of a project. Closely related to change are several other factors that drive the need for CM. Each of these factors also affects the integrity of the product. These factors are visibility and reproducibility, traceability, and accountability and coordination.

3.1.1 Integrity

Protecting the integrity of product components basically means preventing components from becoming corrupted and erroneous. For example, one situation that CM helps to prevent where integrity is compromised is simultaneous updates in the case where two programmers are trying to modify the same module. With no configuration management procedures or controls in place, the programmer who makes the most recent update will have overwritten the update submitted by the previous programmer without realizing it.

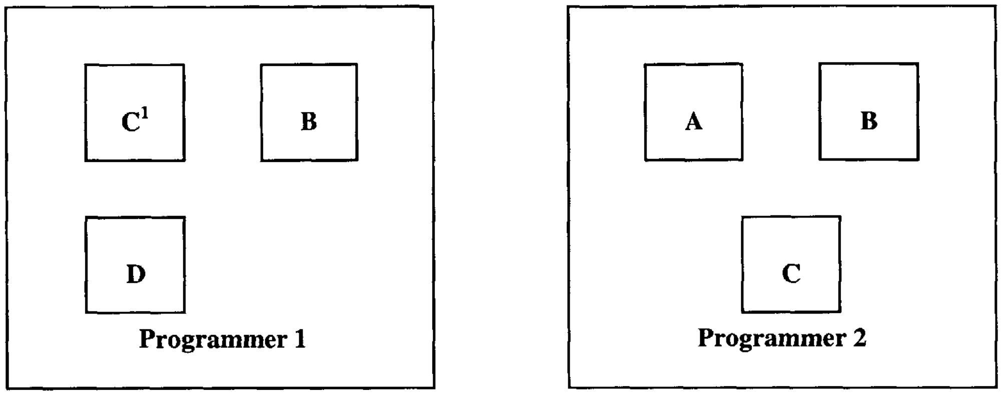


Figure 3-1 Wrong Versions of Same Code (Ben-Menachem, 1994)

Another example of the problem with maintaining the integrity of the product is illustrated in Figure 3-1. This figure shows how Programmers 1 and 2 could have copies of the same piece of problematic code (i.e. code that has a bug). If Programmer 1 fixes the bug without telling Programmer 2, Programmer 1 will be using the correct version, C^1, and Programmer 2 will still be using the unfixed module, C. Along with using an outdated version of the code, Programmer 2 might also change the code so that there are now two different versions of the same module.

3.1.2 Visibility and Reproducibility

Visibility is one of the basic needs in a software project. It means allowing those parties that need to see elements of the product to see it and have access to it. For example, this is important for management purposes, since management needs to be able to observe progress on the product. Visibility control can also facilitate security, preventing those who should not have access to a certain part of the project from having access to it (Ben-Menachem, 1994).

Reproducibility is also a concern. However, the large size of many projects creates the possibility of having many copies a problematic module held by a number of people. If this module later becomes shared by multiple releases of a software customized for different customers, the bug will have to be diagnosed by each person who owns a copy of the problematic code and corrected several times after it surfaces. It is also often of great interest to preserve the elements of a project in such a way that appropriate parts can be reused. This situation necessitates having good storage of project items and descriptions related to each item that will enable others to reproduce and repeat the process.

3.1.3 Traceability

The need for traceability stems from the chaos that can occur when the linkages between project deliverables are not preserved. For example, without appropriate protocols in place, it is very possible to not know what requirements a particular piece of code fulfills. This can become quite a problem in large projects especially (Williamson, 1990). Another problem related to traceability is need for having a project history. Oftentimes a part of a new release may not work properly, requiring that the old version of this part be reinstated. Without tracking the history of the project and its components and storing evolutionary components and keeping the appropriate linkages, it would be impossible to ever restore previous versions of components in case a new version is unsuccessful. A quote from a frustrated software engineer goes as follows: “what do you mean we can’t re-create the previous production version? We must fall back; this one doesn’t work” (Williamson, 1990).

3.1.4 Accountability and Coordination

CM is also very important when there is shared code. In these situations, accountability and coordination are imperative to proper management and preserving the integrity of the product. For example, without having CM protocols are in place, it would be impossible to tell who is responsible for changes made to a project item or who has access to a0 project item. It would also be difficult to know what changes a particular person has made. Without proper accountability, confused software engineers will continue to make statements such as "if we could only find out what changes John made before he went on vacation..."(Williamson, 1990). This situation illustrates the need for accountability in a project.

Coordination is also a driving factor for CM. Without a CM protocol, affected parties might not be notified of changes to project items, such as code, resulting in a breakdown in coordination and an uniformed group. This could result in extraneous or erroneous work. Therefore, there is a need to control versions of the software and track what is under use and under testing or under development.

3.3 Configuration Management Process

Thus, in order accomplish the goal of controlling the evolution of the product, the configuration management process is applied. However, before describing the CM process, a set of key definitions and concepts must first be established.

3.3.1 Key Definitions

Three key concepts that should be defined before discussing configuration management are *Baseline, Change Control Board,* and *Configuration Item.*

3.3.1.1 Baseline

One of the core concepts in CM is the concept of the *baseline.* The baseline is the collection of configuration items for the project defined at particular points in time, often at project milestones and formal technical reviews and approval. These items vary depending on the project, but consist mainly of documents that help to define the project. Before a document is set as part of the baseline, changes to the item may be made easily and informally. However, once the document becomes part of the baseline by approval or reaching of a milestone, a formal procedure must be performed in order to change it (Pressman, 1997).

The project baseline can be divided into four sub-baselines: the functional, allocated, developmental configuration, and product baselines. The functional baseline consists of approved documentation of functional characteristics of the system as a whole and is normally established at the time the contract is awarded. The allocated baseline consists of the approved documentation of functional characteristics of the software such as the requirement specifications. The developmental configuration baseline consists of documents developed during the design, coding and testing of the product. These documents might include design specifications and source code, not to mention other documents. Therefore, this baseline is most likely the baseline that will experience the most changes during the evolution of the product. Thus, although CM is a continual process, the CM team will generally have the most activity during this developmental period. The final sub-baseline is the product baseline. Established only on the completion of testing, this baseline replaces all the other baselines and is the final baseline. It includes the final code on electronic media, user manuals, and other items that are needed for reproduction and maintenance of code (Buckley, 1993).

3.3.1.2 Change Control Board (CCB)

Changes to the project baselines can be made only after being evaluated by the Change Control Board (CCB). The CCB is the central authority that evaluates the requests for changes and approves or denies these requests. The membership of the CCB can vary depending on the complexity and needs of the project. It can be made up of one individual, generally the project manager, or a group of people, including members from requirements analysis, design, testing, quality assurance, CM, the end user, etc. (IEEE, 1990). The CCB meets at regular intervals and holds other meetings when necessary. Regular meetings are attended by all members of the CCB, and other meetings are attended by relevant members.

3.3.1.3 Configuration Item (CI)

The documents and parts of the project that are controlled by the configuration management process and make up the project baselines are referred to as Configuration Items (CIs). The identification of these CIs will be discussed in greater detail later in this document.

3.3.2 Process Overview

The actual configuration management process is centered mainly around the CM activity of configuration control, which will be discussed in further detail in Section 3.4. It is difficult to describe as the CM process as one single process. It is in fact a combination of several processes occurring in parallel during a software project. Therefore it is more useful to describe facets of this process. One aspect of the CM process is configuration item control and how CIs are handled and managed. Another aspect of the CM process is

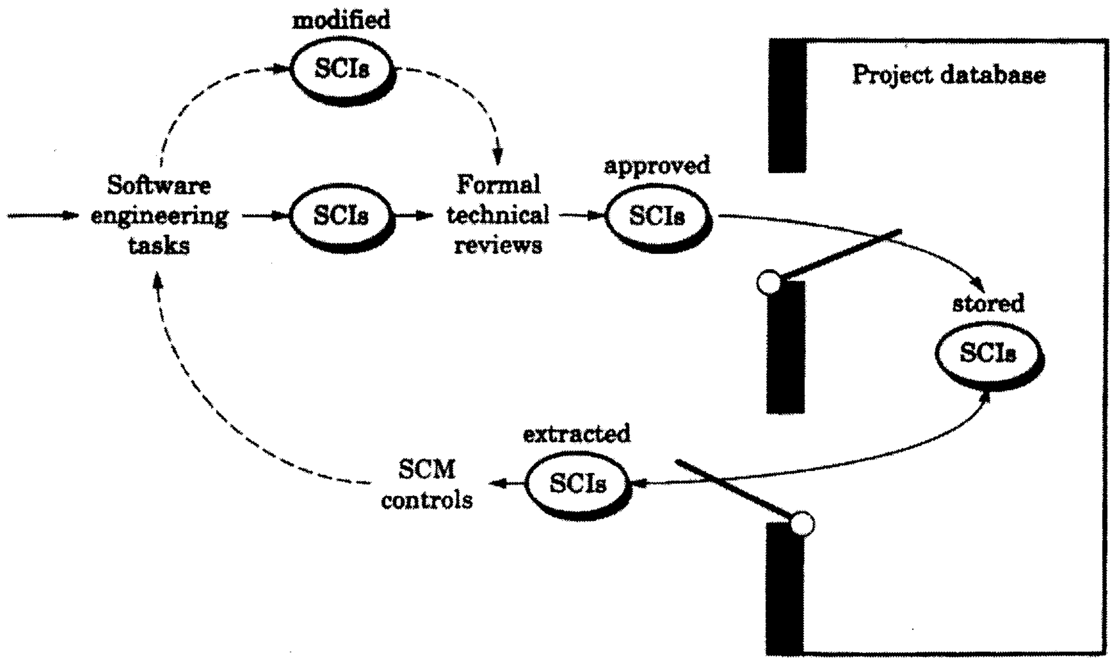


Figure 3-2 Configuration Management Process (Pressman, 1997)

change and how change is managed. The basic CM process from the perspective of CI control is illustrated by analogy in Figure 3-2 and is as follows. CM stores all baselined configuration items in a repository. This repository is represented in Figure 3-2 by the "Project database," and CIs are referred to as SCIs (Software Configuration Items). When modification on a particular baselined CI is necessary, the CI is extracted from the repository. Configuration management controls, such as access controls, are represented as swinging doors in the figure. After the CI is extracted from the project repository, it undergoes various software engineering tasks, resulting in possible modifications to the CI. These modifications can be made only by going through a formal change process, which will be discussed in more detail in Section 3.4. Briefly, this process involves requesting a change, receiving approval of the change, implementing the change, and a review of the change. After these activities, the CI goes through formal technical reviews, such as quality assurance reviews, and after it is approved, the CI is stored back in the project database as a new addition or modification to the baseline.

3.4. Configuration Management Activities

The above process leads to the definition of four main CM activities: configuration identification, configuration control, configuration status accounting, and configuration audits.

3.4.1 Configuration Identification

According to IEEE standards, configuration identification activities "identify, name, and describe the documented physical and functional characteristics of the...elements to be controlled for the project" (IEEE, 1990). These controlled elements are the configuration items. Configuration identification helps to answer the following questions (Williamson, 1990):

- What is this item?
- What application is it part of?
- What documents support and explain the functionality of this item?
- How is this version different from the prior version?
- If this item is changed, what else will have to be changed to prevent problems from occurring?

3.4.1.1 Identifying Configuration Items

In identifying CIs, controlled items must at a minimum consist of the following:

- code
- documentation
- specifications and designs
- test designs, data, and reports
- change reports

In addition to indicating what items are CIs, the configuration identification activity also provides a name and label for each CI, which will aid in versioning, and a description of the item (Ben-Menachem, 1994).

3.4.1.2 Storing and Acquiring Configuration Items

The storage of CIs can vary from project to project. However, CIs will generally be stored in a project repository. There are various ways to store document items because of the different applications that may be necessary to view them, and there are even various ways to store source code. However, all repositories should serve as a central location for items. Having a repository that serves as a central location where CIs are kept allows all members on the project to access the exact same version of a document. This repository has the following functions (Paulk et al, 1993):

- Provide for storage and retrieval of configuration items
- Allow sharing and transfer of CIs between relevant groups
- Provide for storage and recovery of archived versions of configuration items

- Ensure proper creation of products from baseline
- Provide storage, update, and retrieval of CM records

The process of acquiring CIs from the project repository is one that should be carefully monitored and controlled. It may be controlled manually or automatically. For example, according to Williamson, in 1990 the method of one organization was to store master copies of code on disks in a centrally located file cabinet. When a programmer needed to check out code, he/she would remove the appropriate disk from the file cabinet and leave a card with a name and date in the disk's place. In more recent years, it has become common to store items such as source code on a central server. The acquisition of configuration items can then be automated using a software tool. The software tool serves as an "electronic librarian," checking code in and out at the request of the user (Williamson, 1990).

Access privileges to this project repository will be defined differently for each project. For example, for a certain project, programmers may have both "check-in" and "check-out" access to certain code modules and only "check-out" access to others. These restrictions serve as a method of controlling the scope of access. Allowing all users to have full and often unnecessary access to all configuration items could cause great confusion when configuration items are improperly handled, and traceability and accountability would be extremely difficult. Allowing limited access to CIs also serves as a security measure.

Furthermore, how tightly the configuration items are controlled will differ from project to project. Basic access control such as the manual file-cabinet system described above is very tight and active, requiring a great deal of surveillance. When control is very tight, after a CI is checked out, it is basically locked. That is, there is no other access to it until the item is returned to the repository. Automated (computerized) systems make this process easier. However, the advancement of technology and distributed systems has created other needs for access control, as will be discussed in the next chapter.

The acquisition of CIs in general is not to be confused with the process described in Figure 3-2. Along with tracking baselined CIs, CM is also concerned with the tracking and storage of preliminary versions of CIs that will be baselined in the future. Figure 3-2 presents an analogy for the retrieval of baselined configuration items that occurs when changes need to be made to the baseline or when a new CI needs to be added to the baseline. Though the baselined and non-baselined CIs are part of the same repository, before a CI has been baselined, it can undergo changes easily and informally. It does not have to go through the change process described. However, once a CI is added to the baseline, it must go through the process described earlier in Section 3.3.2.

3.4.2 Configuration Control

Configuration control is the core of configuration management. The configuration control process involves two main sub-processes that are very closely related, change control and version control.

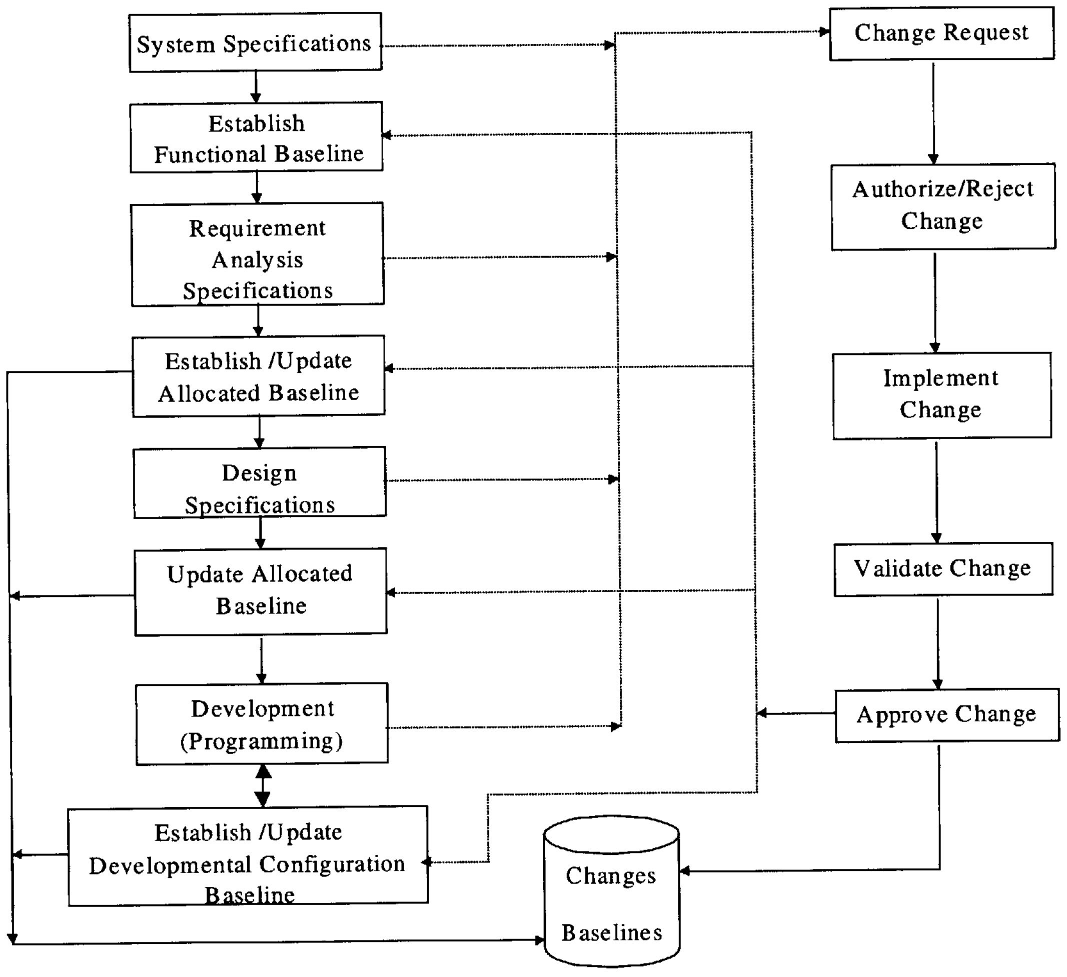
3.4.2.1 Change control.

One of the ways in which CM helps to manage risk is by providing methods for controlling change. Change control helps to answer questions such as:

- "What changes have been made to my software?"
- "How do I inform others of changes I have made to the software?"
- "Do anyone else's changes affect my software?"
- "What is the status of changes I have requested?"

An overview of the change control process is shown in Figure 3-3. As shown in this figure, change can occur at any point in the project. However, this formal change process need only be applied to configuration items that have already been baselined. The left column represents a basic software development process or life cycle such as the linear sequential model. This figure can also be applied to other process models such as spiral or incremental models, which involve several iterations of the linear sequential model.

As the project progresses through the different software development steps, changes can be requested and addressed along the way. There are several steps in the change control process. They are as follows:



System Specifications
Establish Functional Baseline
Requirement Analysis Specifications
Establish /Update Allocated Baseline
Design Specifications
Update Allocated Baseline
Development (Programming)
Establish /Update Developmental Configuration Baseline
Changes
Baselines
Change Request
Authorize/Reject Change
Implement Change
Validate Change
Approve Change

Figure 3-3 CM Change Control Process (modified from Liu and Mantena, 2000)

Change Request

The change request is a key item in traditional, tight change control. The change process must first be initiated by a change request. Valid reasons for changes may include errors or problems discovered in the product, as well as modifications or enhancements to the product. Items that might be affected include specifications, design, and code, to name a few examples. Each project should have a specified procedure established to document change requests. For example, the minimum information that should be provided on a change request is as follows (IEEE, 1990):

- Name(s) and version(s) of the CIs where change is proposed to occur
- Originator's name and group/team
- Date of request
- Indication of urgency
- The need for the change
- Description of the requested change

After a change request has been generated, it is submitted to a central authority, the Change Control Board for review.

Change Approval/Denial

The CCB meets and reviews and evaluates the change request and makes a decision on whether to approve or deny the request for change. Generally, the CCB will review requests generated by problem reports before requests for modifications/enhancements. When reviewing a request, the CCB considers the following information: size, alternatives, complexity, schedule, tests, resources, impact on system, benefits, politics (Favela, 1998), among other possible factors for whether the request should be approved. Based on these factors, the CCB will either approve or deny the change request. After the CCB reaches a decision, it notifies the originator of the request about the decision. If the request is approved, the change is submitted for implementation. The change request also then enters CM's status accounting process. If the change request is denied, the change request originator receives an explanation of why the request was denied along with the notification of denial (Ben-Menachem, 1994).

Change Implementation

Once a change request is approved by the CCB, the next step is implementation. Implementation on the change should be handled through normal channels and by appropriate technical authorities. However, before submitting the change for implementation, the CCB will have determined which changes have higher priorities for implementation. For example, requests for problem fixes will generally have higher priority than enhancements, etc. The CCB may also have to determine what kind of corrective action should be implemented in response to the change request, particularly in the case of a problem report. Once the change has been made, it must be verified by the CCB. Information recorded about the change should include at a minimum:

- The associated change request
- Names and versions of the affected items
- Verification date and responsible party
- Release or installation date and responsible party
- the identifier of the new version

Baseline Update

After the change process has been completed and the change has been implemented, the CM updates the old baseline by incorporating the implemented change into the appropriate baseline, functional, allocated, or developmental configuration baselines, as shown in Figure 3-3. Only changed items approved by the CCB can be added to the baseline. This in turn affects the overall project baseline. The changed baselines are then subjected to quality assurance and testing activities (Pressman, 1997). Finally, the appropriate parties are notified of the changes made.

3.4.2.2 Version Control

Another control process that goes hand in hand with change control is version control. One of the main purposes of CM is to control the evolution of the product and the

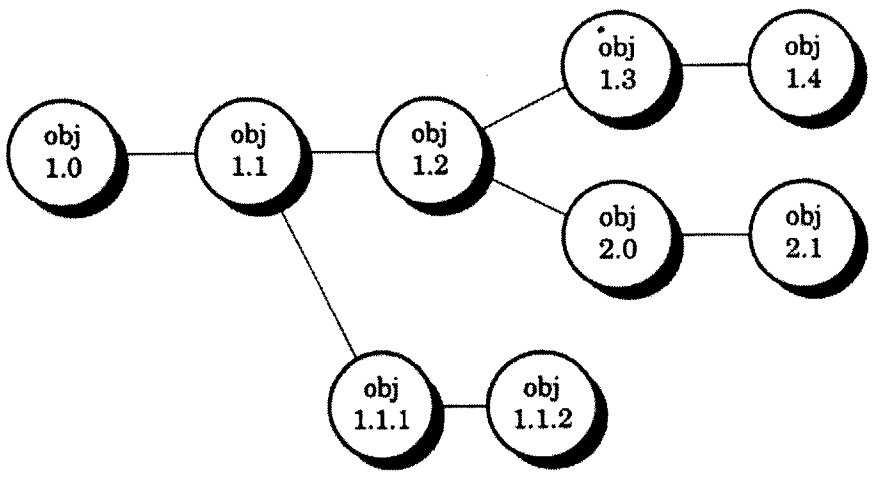


Figure 3-4 Change History of Configuration Items (Pressman, 1997)

associated configuration items during the course of the project. Version control helps to do this by providing a change history for each configuration item. When corrections or changes are made to a CI, a new version of the item is created. Figure 3-4 illustrates this process. For example, if a programmer takes Version 1.0 of a CI and makes a change, the version number becomes Version 1.1. Minor changes to this item would result in version 1.1.1 and so on, and a major change to the CI would result in 1.2. However, a major change that results in a new evolutionary path would be named Version 2.0 (Pressman, 1997).

Another main function of version control is to support the concept of baselining by preventing uncontrolled modification or deletion of configuration items. If a change has been made to a baselined item, this change is noted by changing the version number of the item. Furthermore, by assigning version numbers in tracking the evolution of the CIs, it will always be apparent which version of the CI is the most recent so that changes are not made to the wrong version. As mentioned before, different versions can also be combined to form releases of the product that are tailored for different customers with different features (Willamson, 1990).

Version control is essential to every project. However many methods are available for addressing this need. Simple but rather primitive version control could be sufficient for a very small project. For example, according to Williamson in 1990, one software firm was using index cards pinned to a wall to represent code modules, and cards representing different versions of these modules were pinned below. Every time a programmer made changes to a module, he would add a new index card under the previous one with a new version number written on it. Thus the evolution of the configuration item could be examined by just looking at the hierarchy of the cards. Although this system might work for some projects, however, as software projects become larger, more complex, and more distributed, automated and more sophisticated systems may be required to effectively manage versions. These will be discussed in greater detail later.

3.4.3 Configuration Status Accounting

Configuration status accounting, another configuration management activity, is the recording and reporting of the information needed to manage and physical characteristics of a project and its parts (Buckley, 1993). This activity is particularly important when it comes to changes during the project. Throughout the duration of the project, configuration status accounting helps to answer the following questions about change (Ben-Menachem, 1994, Williamson, 1990):

- What happened?
- When did it happen?
- What were the reasons?
- Who authorized the change?
- Who performed the change?
- What items were affected?
- What is yet to be delivered?

This information is necessary for keeping members of the project informed about the project status, which is especially important in large projects. Keeping members of the project informed prevents many problems, such as two programmers trying to modify the same configuration items with different intent or key people being uniformed about changes that are to be made. This information is made available to various levels of authority via reports. These reports fall into two categories, periodic configuration status accounting reports and reports produced on demand, commonly known as "ad hoc" reports (Pressman, 1997). The periodic status accounting reports list all approved documents that make up the project baselines, as well as the proposed changes, and status of implementation. Ad hoc reports are reports triggered by a problem that has arisen or any other situation where information on CIs is needed (Buckley, 1993).

3.4.4 Configuration Audits

The main purpose of configuration audits is to ensure that change to a baselined configuration item has been properly implemented. This is done using two processes, formal technical reviews and the software configuration audit. Formal technical reviews check the technical accuracy of the modified configuration item, assessing for whether then change made properly reflects the change that was requested, possible side affects, and consistency with other CIs. This review should be done for all changes except the most trivial changes (Pressman, 1997).

The software configuration audit complements the formal technical review by considering the characteristics of a configuration item that are not normally addressed by the technical review. This audit is typically conducted jointly by Quality Assurance and CM teams. The questions addressed by the audit mainly seek to ensure that the change has been made following the proper procedures and are as follows (Pressman, 1997):

- Has the change specified in the change request been made? Have any additional modifications been made?
- Has the change been examined for technical correctness by a technical review?
- Have software engineering standards been properly followed?
- Have the change date and author been specified?

- Have configuration management procedures been followed in making the change?
- Have all affected CIs been appropriately updated?

3.5 Summary

In summary, the main purpose of CM is to preserve the integrity of the products being produced in a project. Along with integrity, visibility, accountability, and traceabliity are also key motivators for CM. In producing a piece of software, there are many activities and tasks that can help in achieving this goal. These activities are the ones that make up the CM process: configuration identification, configuration control, configuration status accounting, and configuration audits. The core activity of these is configuration control.

As business is becoming more global and distributed and collaborative development environments become more common, the software development process has experienced the need to change and adjust to the new environment. CM also has experienced the need to change even as it helps to ease the change. Chapter 4 will discuss the current state of distributed and collaborative CM and how it has dealt with distributed and collaborative environments.

4. Distributed and Collaborative Configuration Management

As software development environments have become more distributed and collaborative, configuration management has been more important than ever, dealing with problems and challenges to evolve and support this environment (Allen et al, 1995). Distributed configuration management has been defined as "simply a recognition of the true state of software development in the 1990's – managing the evolution of software produced by geographically distributed teams, working semi-autonomously, but sharing a common software base" (MacKay, 1995). Distributed and *collaborative* configuration management relies on the same principle but with an added emphasis on collaboration. As the quote suggests, at present, distributed teams are redefining the way people work and have even become quite commonplace, and collaborative efforts are also having a profound impact. Collaboration, fueled by distributive technologies is enabling more large-scaled development projects than ever before. As the product grows in terms of number of developers, size of applications, and cost of product failure, the need for good software configuration management strategy increases (Gumaste et al, 1996). Therefore, it is important to identify and address the CM challenges that come with these new situations.

Though CM faces many challenges in distributed and collaborative environments, the present state of configuration management in distributed and collaborative environments is encouraging. Addressing these challenges has not required much change in actual CM processes, though the way in which these processes are carried out have changed in considerable ways. The most significant changes have been technology-related, as many tools have emerged to assist in the distributed CM process.

4.1 Challenges of Distributed and Collaborative Configuration Management

Many of the challenges that configuration management faces because of increased collaboration and greater distribution are similar to the challenges faced by other processes in the same situation. For example, distributed and collaborative CM deal with CM management challenges, as well as technical challenges, just as other processes in the same circumstances.

4.1.1 Managerial Challenges

As with any distributed project that involves the cooperation of many people, many of these challenges faced are project management and organizational issues. For example, with a large physical distance separating members on the project, how can effective communication be facilitated within the configuration management team? The first difficulty to overcome is coordination problems. Many configuration management tasks, such as CCB decisions on change requests, involve collaboration and group decisions. However, coordinating a large group of people who work in places all over the world requires some effort. For example, distributed and collaborative CM faces decisions such as how its meetings should be conducted over long distances. Timing is difficult, as are possible language barriers and other logistical issues. Although real-time long-distance communication is possible by such technologies as "chat" programs, Microsoft's "NetMeeting," and e-mail, technical difficulties still make these forms of communication and collaboration imperfect at best.

Other managerial challenges for CM include "people" issues, such as how to keep people accountable for their work. When teams are very distributed and have little "face to face" contact, accountability and the feeling of responsibility may decrease for team members. This phenomenon would affect activities not only within the CM team but CM activities that involve other members of the software development team.

In the case where the development team is made up of members from several different organizations, another challenge that project management will face is how to set project standards. Although each organization may already have certain customized procedures for certain tasks or CM procedure, it is necessary to set standards for a project so that there is no confusion about how to perform certain procedures. On the other hand, it is also important to make sure that these standards do not deviate so greatly from an organization's individual system that it cannot be followed without major difficulty.

4.1.2 Process-Oriented Challenges

In addition to the above-mentioned challenges, distributed and collaborative CM also must deal with challenges that are related to the CM process itself. One important issue to consider is the size and efficiency of the CM process. For software development projects, size and level of distribution are directly related. The more distributed a project is, the larger the project in general, and the larger the project, the more opportunities for distribution. Such large size over a distributed area often leads to configuration management in a project becoming unwieldy, hard to handle, and inefficient. One CM process that would be particularly affected is the configuration control process, which includes change and version control. The concern is that the distributed and concurrent development would inadvertently cause interference with others' work. The larger the scope of the project and the more lines of code generated that are interrelated to other parts of the project, the more chance for even small errors to wreak havoc on the project (Ci et al, 1997). Making sure these processes do not become disastrous involves a great deal of manual overhead, which only increases when the project is distributed (Dart, 1990).

The problem of size is often further complicated by another process-related challenge introduced by distributed and collaborative CM, the question of how much bureaucracy and project control to administer to the CM process (Dart, 1992). Tightly controlled CM processes and activities are measures for preventing the activities from becoming undisciplined. Control also ensures the integrity of the system, especially as there are more chances for mistakes and errors in larger systems. However, on the side of the developers, the possible inflexibility, amount of paperwork, and "red tape" in the project resulting from tight control may be frustrating and inefficient and bog the project down (Kliewer, 1998).

Another process-oriented question that CM must answer is the question of how automated the CM process should be. There are many CM tools available that automate (computerize) parts of the CM process. However, the amount of automation these tools provide to the CM process is an issue that needs to be considered.

4.1.3 Technical Challenges

The distributed aspect of distributed and collaborative software development is probably the greatest source of technological challenges for configuration management. When developers are located in different geographical locations, it is necessary to use geographically remote systems to develop the software. However, this requirement presents some problems, especially when the distributed locations are located at great distances from each other. Although technology has significantly improved in recent years so that slow connections and expensive lines are no longer a problem when sites are physically near each other, long-distance communication, such as communication between different continents is often complicated by poor network connections (Allen et al, 1995). Therefore, when planning such a project, it is important to carefully consider the CM needs and what resources are available for addressing these needs.

Another technical challenge faced by distributed and collaborative CM is how administer accessibility to configuration items. This problem is twofold. The first issue is how to provide distributed accessibility for all necessary users at all geographical locations. The question that needs to be answered when addressing this issue is whether CM items are to be distributed or whether developers need only distributed access to a central CM repository that holds CIs (Dart, 1992). The second issue that arises is how this accessibility ought to be managed. In a distributed environment, people are separated physically, making it very no longer feasible to manually manage configuration item accessibility. For example, the "file cabinet" method is not longer valid for managing configuration items. CI management must therefore involve a certain degree of technology. Fortunately, many tools are available for the automation of the configuration.

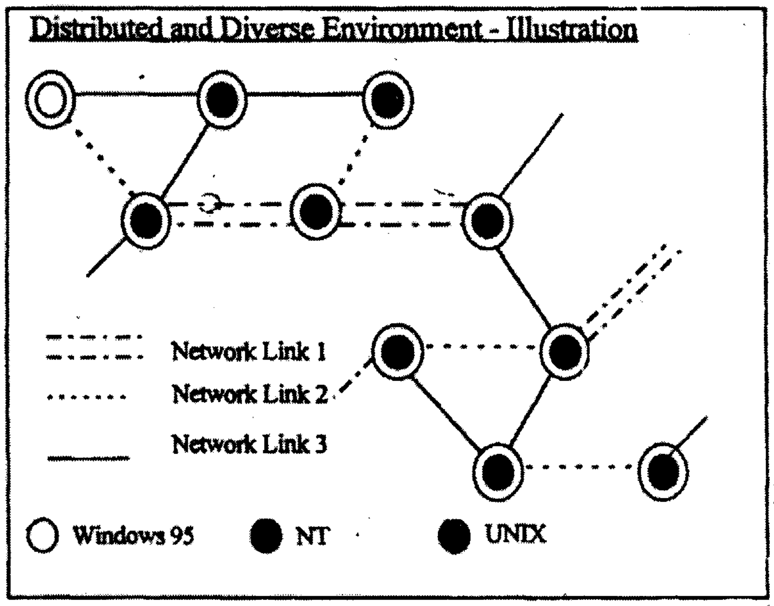


Figure 4-1 Illustration of a Distributed and Diverse Environment (Gumaste et al, 1996)

Distributed and collaborative CM also faces network difficulties, since most networks are heterogeneous, made up of different components that interoperate. Working in a heterogeneous network means that the computing environments of users on the network can be very diverse, utilizing products from multiple vendors and running on different platforms (Gumaste et al, 1996). Figure 4-1 illustrates such a distributed and heterogeneous network. For example, the configuration management network connecting the geographically distributed teams is probably made up of several different platforms. In this situation, certain tools may have to be used to make sure all users can access the necessary files across platforms (Kliewer, 1998).

Although the challenges and barriers for distributed and collaborative CM that are described seem quite intimidating, they are not impossible to overcome. They should not be taken lightly though, especially since they will only become more intimidating as the project becomes larger and as more distributed sites are added. Overcoming these challenges will require thought, careful planning, and in many cases, resources.

4.2 The Present State of Distributed and Collaborative Configuration Management

Presently, distributed and collaborative CM can be viewed as being at a low to medium level of complexity, especially since it is a relatively new development (van der Hoek et al, 1996). However, it is also in a state of transition, especially with the rapid rate at which technology is changing and enabling new collaborative CM opportunities. When comparing distributed and collaborative CM to the "conventional" CM that was first established before the advent of distributed networks, it is evident that the two are actually not so different in process. Although the context that surrounds configuration management continues to change, the core process itself and tasks that need to be accomplished have changed little. However, what has changed more is the ways in which the process is *carried out*. The changes that have occurred are more in the area of team organization and management, system architecture, control, and specific techniques for carrying out the CM activities.

4.2.1 CM Team Organization and Management

As the organization of projects changes, so must the organization of configuration management for these projects. Presently, CM teams are organized in both centralized and decentralized fashions. In the case where the project is organized by one organization with collaborating satellite offices, CM teams still remain rather centralized, monitoring and controlling software configuration from a central location. However, CM teams are becoming increasingly decentralized. For example, in cases where collaborating remote locations already have their own CM teams, the CM team is very decentralized. People relevant to CM are scattered around the globe, and it may also be necessary to have a representative from each remote location to aid in the collaboration between remote teams. Thus, decentralization is necessary in these situations.

4.2.2 CM System Architecture

The CM system architecture both influences and is influenced by distributed and collaborative project development environments. First of all, the CM process is a vital part of distributed and collaborative software development (Ci et al, 1997). Therefore, it is important that CM configure the project as well as possible with the tools and software that will be most compatible with the established network and most helpful to the project. For instance, one of the most important choices that CM makes is the selection and design of the project repository. However, there are also many external factors that are beyond the control of configuration management that influence CM. One such example is of the system architecture that CM must work around is the established network configuration.

4.2.2.1 CM Repository

At present there are a few main methods for the storage and acquisition of software configuration items for a distributed and collaborative software engineering environment. One method being used for configuration item storage and retrieval today is a central global repository (Allen et al, 1995). Using this method, all users have access to the central repository across a Wide Area Network through client/server relationships to the repository (van der Hoek, 1996). For example, a user in California would be able to access and update a repository located in Massachusetts. Documents and code are available for users to withdraw from the system and work on, and these items are then checked back in when the work has been finished, creating a new version. Since the repository resides on one node, from the computer's point of view, all users have equal access, accessing the repository using the same protocol and a standard shared file system (MacKay, 1995). This method operates using the atomic transaction method. This means that a file cannot be checked out by more than one person at a time to ensure that simultaneous changes cannot be made on the same file (Dart, 1990). This method is typical for the example of a software development project where there is one organization, and control is centralized at one place, with all other nodes on the network considered satellite offices. Since all the teams represent only one organization, it is possible to allow shared access and enforce standard protocols.

Another way in which CIs are stored is referred to as the distributed components method. This method also provides a central project repository; however, it allows the CIs to be distributed to remote locations, and transparently (Dart, 1990). When a user needs to access a CI, a snapshot or copy of the source is taken and sent to the remote locations. Any changes that are made to configuration items are made at copy at the remote location and then carefully merged back into the sources at the master site (Allen et al, 1995). This is done using a CM tool, FTP or even the surface mailing of magnetic tapes (Belanger, 1996).

4.2.2.2 CM/Computing Network

As technology has progressed and computing environments for software development have changed, the homogeneous environment has become obsolete. The current standard is the heterogeneous distributed environment mentioned earlier, which connects distributed environments using a loosely coupled network (Gumaste et al, 1996). This heterogeneous computing network also influences configuration management, as configuration management must now also be performed over a heterogeneous network.

For instance, it is quite common for developers in remote locations to develop on different platforms. However, the mixed platform networks make CM more difficult in some ways. For example, in a mixed platform network, accessing CI repositories becomes a challenge. In most cases, it is helpful to use a CM tool to aid in file handling and security. The following is a list of four variations on handling a mixed platform network with respect to configuration management. These variations are based on the presence of a CM tool and a Network File System (NFS), a client/server application which allows users to view, write, or store files on a remote server as if it were the client's own computer (Kliewer, 1998).

- No client CM tool and no NFS. In this situation, developers must telnet to the repository and use a protocol such as FTP to move files back and forth.
- No client CM tool, NFS available. In this situation, developers must still log into the repository host. However, since there is an NFS, files can be accessed without copying back and forth.
- No client tool, but NFS available. Because there is no NFS, the CM tool must find its way to the files in the repository.
- CM tool and NFS available. In this situation, CM is transparent to all users, allowing users check in/out files as if they were on a local disk. This is the most favorable system.

Since most tools run on limited platforms, the configuration of the CM system will influence the choice of tools and features for the project. If certain unsupported tools or features are deemed necessary for the project, the network should be set up accordingly.

4.2.3 CM Process

The CM process is also currently undergoing a transition from traditional process methods to more distributed and collaborative methods. However, the tasks still remain the same; they are only approached and facilitated in new ways. The main activities that are affected are configuration identification and configuration control.

4.2.3.1 Configuration Identification

In many ways, configuration identification for a distributed and collaborative software development project is very straightforward, as the items to be identified are typically the same as in any other development environment, such as the items discussed in Chapter 3. However, at the same time, configuration identification in distributed and collaborative environments at the present, particularly the storage aspect, still experiences many challenges and inadequacies.

One positive aspect of the current method of configuration storage is the visibility it allows to distributed developers. Although the current methods are not without difficulties, they do provide a basic visibility to the distributed development team, be it by providing access to the original item or by providing a replicated copy. In a central repository situation, it is important and now typical to have some way for all configuration items that have been made a part of the system to be made visible to an authority, usually by using some kind of CM tool. In the same way, visibility of a CI can be given to defined groups of people who may need to see an item before others (Feiler, 1990).

With respect to the actual identification and tracking of a CI, the challenges are still great. Because developers are located in different areas and may handle different modules, it is often difficult to know where a particular CI is located or who controls a particular CI, especially if the CM repository is decentralized. In the case of decentralization where copies of an item exit at different locations, it is often difficult to determine the original item (Allen et al, 1995). In addition, according to current CM practices, it is up to the owner of the module or CI to put the CI into the control of the CM system. Thus, distributed and collaborative CM is also very dependent on establishing official CM policies and on all members' of the development team knowing and following these policies. Establishing policies are also important because of the collaborative nature of this kind of software development. With so many people in different locations and parts of the organization working together, a particular CI is relevant to a large number of developers. Thus, these policies should are set by a central authority and followed carefully (van der Hoek et al, 1996).

4.2.3.2 Configuration Control

Currently, the CM configuration control for most organizations is fairly tightly controlled. As discussed in an earlier section, having tight control is typical of traditional CM, and as projects become larger and more distributed, organizations are finding it useful to exercise more strict control over CM. Automated tools are aiding in this management, which is difficult to have over somewhat undisciplined networks, offering better control and without the overhead normally associated with greater control (Dart, 1990).

The change control process is still partly manual and currently very controlled. Although change control forms can be submitted to the appropriate parties via e-mail and approval can be given in the same way. However, some CM tools take change control into account by locking baselined CIs so that changes cannot be made without approval. Version control also benefits from the extra measures of control offered by some automated CM tools. Using these tools, a history of a configuration item over the evolution of the product is kept, as well as the version number of the CI and its difference from the previous version.

One difference from traditional non-distributed CM is the almost inevitable decentralization of the Change Control Board. However, even though the CCB is physically separated, it generally still functions as one unit, using of e-mail and other electronic forms of communication to keep communication lines open.

4.2.3.3 Configuration Status Accounting and Configuration Audits

Configuration Status Accounting and Configuration Audits are two CM activities that remain much the same in a distributed and collaborative environment. However, the benefits to CI management, version handling, and control offered by CM tools also do filter down to CM status accounting and audits. For example, the versioning tools enable a history of changes for an CI to be tracked, allowing for easier audits and simpler determination of whether the appropriate measures have been taken in making changes. Tracking the status of change requests is also simpler, since CIs and changes are available for viewing through versioning tools.

4.3 Configuration Management Tools

Tools that aid the configuration management process have become very widespread and virtually indispensable for distributed and collaborative software development environments. These tools generally automate the configuration control process and drastically lessen the amount of overhead that is spent managing configuration items in distributed environments. However, the degree of automation that occurs varies depending on the organization and the tool being used. At present, the CM process for most organizations is a combination of manual and automated procedures (Dart, 1990). For example, the storage and acquisition of CIs, as well as version control, may be automated with a CM tool, but with many tools, change control forms must still be filled out manually to start the change process.

Choosing the right CM tool is often difficult, as there are currently many tools to choose from but few that will be able to meet all the needs of an organization. In many cases, an organization may choose to modify a "shrink-wrapped" CM tool or even develop its own to meet the organization's needs (Dart, 1992). In any case, no shrink-wrapped tool will be able to provide a total configuration management solution in a distributed and diverse environment (Gumaste et al, 1996). Another problem that a project may run into with respect to CM tools is that today's CM tools often cannot run on all the project platforms. The CM tools currently offered can be divided into four categories, each representing a different set of paradigms: Check Out/Check In, Composition, Change Set, and Transaction (Dart, 1992).

4.3.1 Check Out/Check In Model

The Check Out/Check In model is based on the concept of a repository, as discussed in Section 4.3.2.1. The key idea is that users access a central repository that holds all configuration items and check out and check in items for modification, and concurrent modification is facilitated by locking mechanisms. Versioning is independent using this model (Chan and Hung, 1997).

This model has been widely adopted by earlier models of CM tools, such as SCCS (Source Code Control System) and RCS (Revision Control System). SCCS one of the most important first generation tools. A UNIX-based source code revision and version control system, SCCS stores multiple revisions of source code files in an evolutionary tree structure. It is also one of the first systems to incorporate change control into its set of features. RCS is also a UNIX-based tool for configuration management. Organized in a tree structure just as SCCS, RCS assigns versions labels using a numbering scheme. Instead of storing whole files, RCS stores only the differences between versions in the repository (Dart, 1990).

4.3.2 Composition Model

The Composition model improves on the Check In/Check Out model by first building a system model based on an original configuration item to provide information on the composition and structure of the configuration. This information will assist the tool in maintaining the item's integrity in the future (Dart, 1992, Chan and Hung, 1997).

DSEE (Domain Software Engineering Environment) is a good example of a CM tool based on the Composition model. Designed to handle large-scale development projects in a distributed computing environment, DSEE provides the system-building mentioned above and source code control very efficiently over a wide network (Chan and Hung, 1997).

4.3.3 Change Set Model

The Change Set model also utilizes a repository and allows a logical change to a CI and a way to create any version of a configuration without requiring it to be related to the latest version of the item. An example of a tool following this model is ADC (Aide-De-Camp) (Chan and Hung, 1997). Using ADC, after a requested change is approved, the CCB assigns a workspace to the developer. This workspace consists of a set of directories and files, a local copy of the files in the repository. After the developer has completed the changes, these local files are deleted and the changes are updated to the repository (Dart, 1990).

4.3.4 Transaction Model

The Transaction model incorporates repository and workspace concepts, as well as the concept of atomic transactions (Dart, 1992). This model provides support for concurrent changes to configuration items (Chan and Hung, 1997), allowing users to change the same parts of the product.

An example of a tool using this model is NSE (Network Software Environment) (Chan and Hung, 1997). NSE allows developers to work on the same configuration items by providing private workspaces for development and synchronizes the changes made to the parent environment (Chan and Hung, 1997). If developers have made conflicting changes, NSE notifies the developers about the conflicts and assists in solving them (Dart, 1990).

4.4 Summary

In many ways, configuration management of the present is very different than configuration management of the past. Although many challenges have been introduced by distributed and collaborative development environments even, they are at present being sufficiently addressed. The actual CM processes have not changed greatly, but many new technologies and tools now exist to aid CM in a distributed environment.

Thus, collaboration can also be assisted, as distribution and collaboration increase together. Chapter 5 will present a case study on distributed and collaborative CM in order to illustrate the points addressed in this chapter and also suggest some ideas for the future of distributed and collaborative CM.

5. ieCollab - A Case Study for Distributed and Collaborative Configuration Management

Distributed and collaborative software development environments have still yet to really take off. However, many have acknowledged the importance in the future of software engineering and are studying this topic both in the research world and in industry. As mentioned earlier, the methodology of how to conduct CM in this environment has been recognized as having significant impact and is of particular interest to many.

This chapter will describe the ieCollab project as a case study for distributed and collaborative software development, particularly focusing on the role of configuration management in the project. First a project description will be given, followed by an analysis of the project's CM methodology. Finally, this chapter will aim to evaluate problems that arose during the ieCollab project and make recommendations on how the methodology could be improved.

5.1 Project Description

The following section will describe the ieCollab project concept, its background, purpose, development environment, project organization, development process, and project technology.

5.1.1 Background

ieCollab is the abbreviation for "**i**ntelligent **e**lectronic **Collab**oration." This is the name of a software development project that was conducted by students from three universities located around the western hemisphere:

- the Massachusetts Institute of Technology (MIT) in Cambridge, MA, United States
- Pontificia Universidad Catolica (PUC) in Santiago, Chile
- Centro de Investigacion Cientifica y Estudios Superiores de Ensenada (CICESE) in Ensenada, Baja California, Mexico

The forum that allowed this collaboration to occur was the class "Distributed Development of Collaborative Engineering Support Systems" otherwise known as 1.120, a class spearheaded by MIT but taught jointly by the three universities. This alliance was first formed between MIT and CICESE in 1996 and has been ongoing ever since. The results of the first three years of collaboration are three versions of a software product for distributed meeting management and resource sharing called **C**ollaborative **A**gent **I**nteraction and Synch**ro**nization (CAIRO). In 1999, PUC joined MIT and CICESE so that there were now three collaborating universities. Together, these universities conducted the ieCollab project over a five-month period, beginning in November, 1999 and ending in April, 2000.

The ieCollab team was composed of 34 students, with five students from PUC, five from CICESE, and 24 from MIT. During these five months, the students worked in a distributed and yet collaborative manner with the goal of producing a software package for distributed teams that is either based on or inspired by the CAIRO product.

5.1.2 Purpose

The ieCollab project and tool was motivated by the current influence of geographically dispersed and collaborating teams and the challenges that are faced with this kind of working environment. Thus, the purpose of the ieCollab project was two-fold. The first purpose was to develop a product to aid in geographically independent collaboration.

During the course of the project, it was decided that this product would be an Internet-based collaborative application with the following functionality (Abbot et al, 2000):

- Document Sharing
- Application Sharing
- JAVA meeting environment
- Meeting management facilities (provided to other web-portals like Yahoo, and Lycos, as well as its own portal)

The development of this product served to help students meet the second goal of the project, which was to provide a way for students to learn about the software development process in a collaborative and distributed manner. By developing the ieCollab product in a distributed and collaborative environment, students were able to understand both the benefits challenges related to this kind of development environment and were also able to see the need for a product such as ieCollab.

5.1.3 Development Environment

The ieCollab project was developed collaboratively and over a distributed geography, allowing team members to experience both the difficulties and rewards of working in such an environment.

5.1.3.1 Distributed Environment

As all developers worked on the project from their respective campuses, the ieCollab project was a distributed project. As with any distributed project, there was a need for document sharing, and because this was a software development project, there was also a need for resource sharing. These requirements meant that reasonable network connectivity between the three campuses was imperative. MIT, CICESE, and PUC all have their own campus-wide networks, and these networks all connect to other outside networks. This architecture forms a loosely coupled network connecting MIT, CICESE, and PUC.

Thus, the ieCollab project required a certain level of connectivity between campuses and made use of the networks connecting them.

5.1.3.2 Collaborative Environment

As described previously, the idea of collaboration and working together was a key aspect of the ieCollab project. The geographic distribution of the developers also contributed to the need for collaboration. This collaboration occurred in many forms. The first type of collaboration that occurred is the collocated collaboration between developers from the same school. Although software development is inherently rather collaborative, in industry it is often very common for developers to work rather autonomously (Belanger et al, 1996). However, because none of the students in the ieCollab project had received prior education in the software development process, it was difficult for students to work independently and much more useful for them to work collaboratively. Thus, students became experts at their own roles and shared knowledge with others, and in this way, they learned from each other.

With respect to working with the student developers from other universities, collaboration and communication were important in addressing the many logistical, cultural, and technological. For example, there was a great deal of attention paid to communication and teamwork, since developers came from many backgrounds and cultures represented. Dealing with these challenges required heightened and more thoughtful collaboration to ensure that these barriers did not spiral out of hand.

5.1.3.3 Distributed and Collaborative Environment

Working from separate campuses required a considerable amount of collaboration on the part of the team members, since almost every team was made up of developers from more than one location. This combination of dispersed geography and collaborative work presented a bit of a problem.

With respect to working with other universities, collaboration was imperative for any kind of productive work between campuses. Situations where there was a lack of collaboration resulted in a breakdown of communication and over-independence and developers at different campuses becoming isolated, defeat the purpose of the whole class. Thus, distance was an obstacle to collaboration in some ways.

Although the project faced many of these challenges, ieCollab was able to facilitate collaboration over a long distance by utilizing many tools. The most important of these tools is e-mail, since it is free and accessible to everyone. However, this method was not without its difficulties, as the network downtime at CICESE was often a factor. For example, the collaborate.mit.edu web repository was often inaccessible by the CICESE team, and even e-mail was inaccessible to them at times. Tools that allowed immediate and concurrent collaboration such as long distance phone calls, CARIO, ICQ, and NetMeeting were also useful; however, the phone calls were expensive, and ICQ and NetMeeting suffered from the same network downtime. Many of the tools were also logistically difficult to handle, especially since the MIT, PUC, and CICESE are all in different time zones.

5.1.4 Project Organization

The ieCollab team was divided by project roles, which are listed below:

- Project Manager
- Business Manager
- Marketing Manager
- Project Advisor
- Requirements Analyst
- Designer

- Programmer
- Tester
- Quality Assurance
- Configuration Manager
- Knowledge Manager

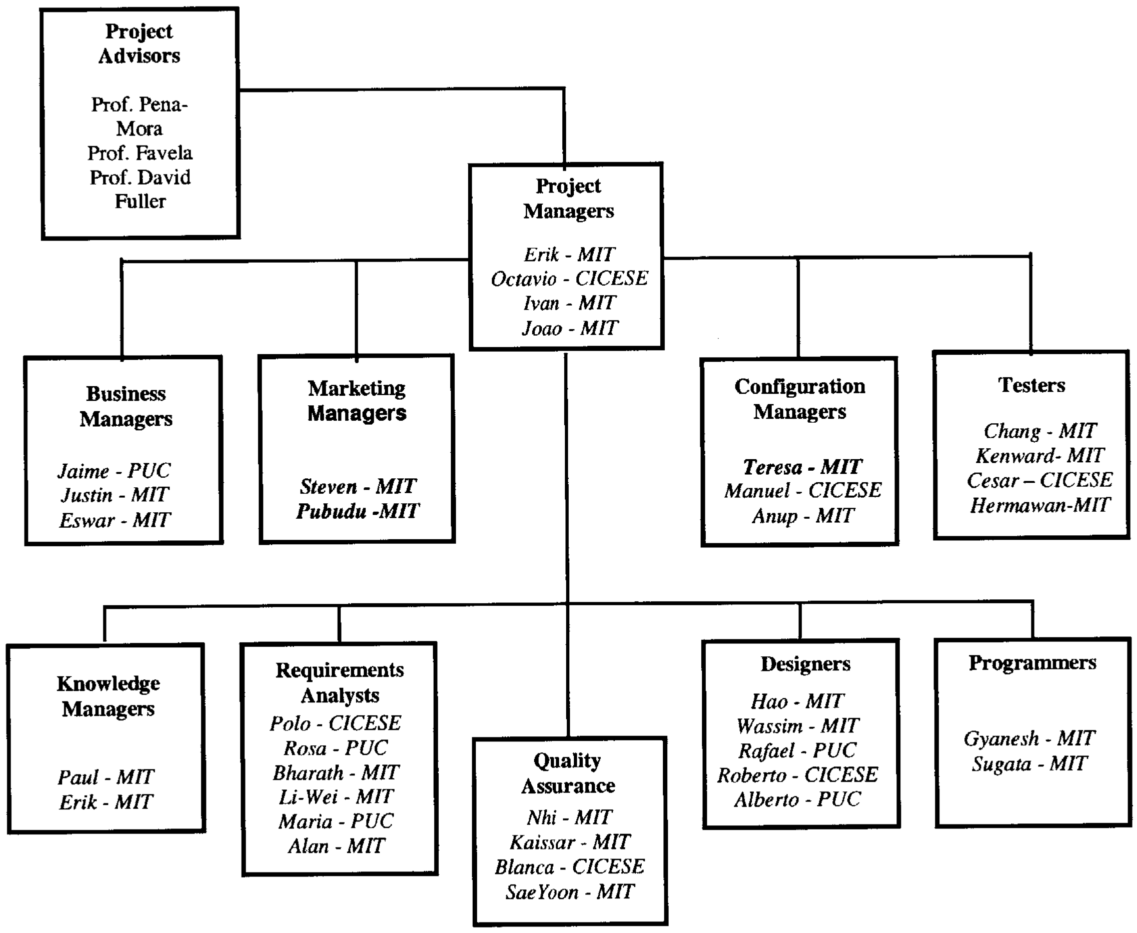


Figure 5-1 ieCollab Organization Chart

The project was organized so that a team of students performed each of these roles, and each team was generally organized to include students from at least two different universities. In addition, virtually all students had more than one role. The ieCollab organization chart is shown in Figure 5-1.

5.1.5 Development Process

The development process for ieCollab followed typical development techniques as taught by the 1.120 instructors and followed by the industry. Most of the project roles listed above are representative of the roles that developers take in a typical software development project. However, these roles for ieCollab were customized for the project. In some cases, new roles were created. These roles were incorporated into the entire software development process of ieCollab, which also followed software development industry standards (IEEE, 1990).

5.1.5.1 Roles

Project Manager

The ieCollab project manager (PM) team's responsibility was to handle the activities associated with the project management umbrella activities discussed in Chapter 2 of this thesis. For ieCollab, these included project organization, planning, monitoring and control, as well as resource allocation among the teams. Furthermore, the PM team was responsible for giving necessary support when needed (Abbott et al, 2000). This support included conflict resolution, aiding in collaboration, delegation of responsibilities, and allocation of resources, among others.

Business Manager

The business managers' main tasks correspond with the tasks for what is typically known as the system engineering process. In the ieCollab project, the business manager team was responsible for determining the market needs and defining the goals of the ieCollab product. Working closely with the marketing manger team, the business mangers also talked to the customers to identify the features that the product should have (Abbott et al, 2000).

Marketing Manager

In line with the system engineering process, the ieCollab marketing managers were responsible for identifying the market characteristics, technology trends, and customer needs, as well as competitors in the market. Using this information, they developed a competitive strategy for what kind of features the product should have in order to be competitive and address market demands and how the product should be positioned, among other issues (Abbott et al, 2000).

Project Advisor

The project advisor role in the ieCollab project was a faculty advisor/professor who helped to supervise the project and provide input to the business and marketing managers. The project advisors could be viewed in a way as experts on the market in which the ieCollab product would be introduced.

Requirement Analyst

The ieCollab requirement analyst (RA) team was responsible for taking the business manager team's goals and translating them into software requirements. Using various methods such as use cases, the RA team specified the desired functionality of the software package, describing the interfaces with other systems and design constraints in such a way that it could be translated into design terms (Abbott et al, 2000).

Designer

The main responsibility of the ieCollab design (DE) team was to translate the requirements of the software defined by the RA team in such a way that the programming team would be able to comprehend and turn into actual product code. This information was also of interest to the testers for determining whether these design functions had been met at the end of the project (Abbott et al, 2000).

Programmer

The role of the programming (PR) team in ieCollab was component implementation, as described earlier in Chapter 1. The programmers strove to create well-documented and quality software code for the ieCollab product that followed the design plan set forth by the design team (Abbott et al, 2000).

Tester

The main goal of the ieCollab testing (TE) team was to verify the proper operation of the product and to identify errors (Abbott et al, 2000) to ensure that what was delivered to the "customer" was a quality product, able to perform what it professed to perform, and as error-free as possible.

Quality Assurance

The ieCollab quality assurance (QA) team's main tasks were to conduct various activities such as inspections, walkthroughs and audits in order to ensure that the proper standards were followed during the creation of the product and that what was produced was a quality product (Abbott et al, 2000).

Configuration Manager

The ieCollab CM team's main responsibility was to control the evolution of the products generated in the project and to inform the necessary team members of the software status, including changes, versions, and storage. This helped in preventing simultaneous updates and informing those involved of common code modifications. This ieCollab CM process will be discussed in further detail later in this chapter (Abbott et al, 2000).

Knowledge Manager

The ieCollab knowledge management (KM) team was responsible for maintaining the project web site (www.collaborate.mit.edu) and defining the standard format for project documents, as well as producing the User and Technical Manuals (Abbott et al, 2000).

Interestingly, KM is not a typical software process. However, it is a discipline used by many organizations for acquiring and controlling knowledge within the organization. However, since ieCollab was an educational endeavor, KM was introduced to pay special attention to the memory of the project for future years (Wong, 2000). In fact, KM in ieCollab took on some responsibilities of a typical CM team by controlling the collaborate.mit.edu site, which tracked the evolution of non-code documents. ieCollab's KM team also took on some responsibilities of the QA team by defining standard document formats.

5.1.5.2 Process Models

The ieCollab team initially planned the project according to the waterfall process model. Please refer to Figure 2-3 for an illustration of the waterfall model. Thus, each step was to be completed before moving on to the next process. For example, the design process would not begin until the RA process was completely finished, programming would not begin until the design was complete, and so on. According to this waterfall process model, at the end, the finished product would be rolled out all at once. However, the process model for the project was modified halfway through the project along a modification to the project scope. After this modification, the ieCollab product was developed according the incremental model for software development, planning to release two modules of the product at separate times. The incremental model is illustrated in Figure 2-4. The first module, Version 1, described as the "Meeting Management Environment," was to allow distributed users to set up and manage on line meetings (Abbott et al, 2000). The second module was to be released a few weeks later. Named "Version 2-Transaction Management," this module was to allow the ieCollab server to track usage of ieCollab's meeting management service and charge fees on per-transaction basis (Abbott et al, 2000).

Splitting the project up into two modules was both helpful and detrimental to the goals of the ieCollab project. The incremental model was helpful to the project delivery, since the overlapping work in the incremental model allows for better use of time and resources, and time and resources were two particularly challenging areas in the project because of the project's educational nature. It terms of resource allocation, the modular aspect of the project allowed the parts to be divided between the different universities so that it was easier for developers to develop within themselves at their own universities. However, this division, was, in a way, detrimental to the collaborative nature of the project. For instance, toward the end of the project, the project was so modular that the two campuses still involved in the project at that point became rather isolated from one another.

5.1.6 Project Technology

To create the ieCollab product, the development team made use of many technologies and tools. For example, the programming language used was the Java language, JDK 1.2.2 to be exact, which was very useful for both distributed and collaborative natures of the project. Since Java is platform-independent, this allowed all developers to participate in the code generation independent of where they were located or what platform their university used. Being able to use Java on any platform also made collaboration easier. Also used was JDBC, or Java Database Connectivity, an interface that allows the Java language to interface with databases (Chen et al, 2000).
Another key technology used in the project includes CORBA (Common Object Request Broker Architecture) (Chen et al, 2000). CORBA is an architecture and specification for creating, distributing, and managing distributed program objects in a network. This makes software development much easier and provides clear interfaces for other applications. CORBA was necessary for the ieCollab clients to access the ieCollab server (Chen et al, 2000).

ASP (Application Service Provider) architecture was another technology incorporated into the ieCollab product. ASP architecture allows organizations to access applications that are not stored on their own enterprise servers but a special kind of application server that is designed to interact with stripped-down thin client workstations. The ieCollab server was to be an ASP server supporting the meeting management feature of the ieCollab product, which was designed with distributed collaboration in mind (whatis.com, 2000).

5.2 ieCollab Configuration Management Overview

The following chapter gives an overview of the configuration management role in the ieCollab project, including its organization, challenges faced, responsibilities, and processes.

5.2.1 ieCollab CM Team Organization

As virtually every other team in the ieCollab project, the CM team was made up of students from more than one university. Table 5-1 shows the organization of the CM team. The team consisted of two students from MIT and one from Mexico. Since the majority of the team members were located at MIT, the team leadership was at MIT. However, there was still quite a bit of opportunity for collaborative work.

Table 5-1 CM Team Members (Liu et al, 2000)

Title	Name	Location
Team Leader	Teresa Liu	MIT
Team Member	Anup Mantena	MIT
Team Member	Manuel Alba	CICESE

The Change Control Board, though a separate entity from the CM team, was also an essential part of the Configuration Management process. The CCB also consisted of almost all MIT students. Table 5-2 indicates breakdown of team members on the CCB. The reason for this heavy imbalance in the representation on the CCB is that project management defined the CCB as consisting of the team leaders from each team. Since MIT students generally made up the vast majority of students in the ieCollab project, the team leaders were concentrated at MIT. This organization of CCB members was helpful in the case CCB meetings needed to be held, as it was fairly easy to arrange a meeting time between the board members at MIT and one member in Mexico. However, PUC was not represented at all in the CCB. This most likely did not present much of a problem in the project, since the CCB met infrequently at most. If the project had gotten an earlier start and if the CCB had needed to meet more frequently, on the other hand, this lack of representation might have affected collaboration and awareness with respect to PUC.

Table 5-2 ieCollab Change Control Board (Liu et al, 2000)

Team	Team Leader
Project Manager	Joao- MIT
Business Manager	Justin-MIT
Marketing Manager	Steve-MIT
Requirement Analyst	Polo-CICESE
Designer	Hao-MIT
Programmer	Gyanesh-MIT
Quality Assurance	Nhi-MIT
Tester	Kenward-MIT
Configuration Manager	Teresa-MIT
Knowledge Manager	Paul-MIT

5.2.2 ieCollab CM Responsibilities

The ieCollab CM responsibilities were for the most part very similar to the general CM responsibilities for any software development project that were described in Chapter 3. However, unlike most projects, ieCollab CM had a varied amount of control over some configuration items. This situation occurred because the storage of certain CIs, namely non-code CIs, such as project specifications and testing documents, was also to be controlled by the knowledge management team. This situation was necessary since one of the objectives of KM was to record the evolution of project documents for historical purposes and future use. However, it was still the responsibility of CM to control these documents once they had been approved and made part of the project's baseline. Thus, CM and KM shared responsibility of non-code configuration items, and CM had complete responsibility over controlling source code.

Thus, after developing a plan for configuration management (please refer to Appendix to view the CM Plan), the CM team was responsible for setting up and managing a system for managing the evolution of project configuration items. These tasks included: creating a repository for CI storage, ensuring distributed access to it, setting standards and protocols for accessing and changing configuration items, tracking and versioning them, and reporting the status of the items, to ensure that these items were maintained with integrity throughout that evolution of the product. This was to be done by carrying out the four main CM activities, which will be discussed in greater detail later in this chapter. The CM team also had to perform other project activities that were unrelated to CM work.

Within the CM team, team members had different responsibilities:

CM Team leader

- Team management
- Create CM Plan
- Submit weekly reports to PM
- Serve as CCB liaison
- Aid in CM activities

CM Team members

- CM activities
- Help to create CM Plan
- Update CM leader on weekly progress

Each member of the CM team had different levels and areas of expertise; thus, different team members were more involved in some activities than others. However, throughout the project, the CM team strove to conduct their work in a collaborative way by trying to have each member be involved in all activities to a certain degree, even if it was only by providing input and suggestions.

5.2.3 CM System Architecture

As predicted in Chapter 3, the system architecture of the ieCollab project was determined by a combination of choice and circumstance. The architecture chosen for the CM repository and network are described below.

5.2.3.1 CM Repository

The repository that the ieCollab CM team chose for the code CIs in the project is Concurrent Versions System, better known as CVS. Using CVS, the ieCollab team set up a central project repository on an MIT UNIX server, cee-ta.mit.edu. This repository operated on a central repository provided local copies such that developers from any campus could access configuration items on the central server and make copies onto their local workstations to make changes. These changes did not affect the "original" CI, nor did they affect the local copies of other developers. Once the changes were made, developers then committed them back into the repository, which made the changes on the server. Access to this repository was given only to those who required access, mostly programmers, testers, QA, and others who needed to view code. Other aspects of the CVS tool will be discussed in more detail later in this Section 5.4.1 of this chapter.

There were several advantages realized by using this type of repository for the ieCollab project. One advantage is the flexibility of checking out CIs. For example, utilizing a global repository where only one developer could work on or even look at a file at a time would have severely affected the efficiency of the project. Along the same line of thought, the visibility of configuration items using local copies aided in the collaboration with team members. Many members of the programming team were working together on the same Java classes, and allowing them visibility to the same files was almost necessary for feedback and collaboration between team members. Using an atomic transaction-based repository would have hindered this kind of collaboration, since only one instance of the file could exist and be viewed. Allowing multiple visibility also aided in the learning process, helping to satisfy one of the project goals.

CM also created a separate repository for non-code configuration items. This was done as a joint effort with the knowledge management team. This repository, called the "CM/KM Web Repository," held all the approved and latest non-code CIs such as RA specifications, testing documents, and other CIs. Although the CVS repository could have handled these CIs, the CM team chose to create the web repository on-line for easy viewing of these CIs (Liu et al, 2000). The CM/KM Web Repository will also be discussed in more detail later in this chapter.

5.2.3.2 CM Network

The ieCollab CM network was a loosely connected, mixed-platform network. The main operating system at MIT is UNIX. This UNIX environment was accessed either directly by workstations or indirectly by telnet from a WindowsNT environment. PUC and CICESE are also supported on UNIX and Windows 95/98/NT operating systems. As the repository was located at MIT on the cee-ta.mit.edu server, PUC and CICESE used telnet technology to access MIT's UNIX environment from their respective locations. However, PUC and CICESE's access was limited to the cee-ta.mit.edu server and did not include access to the MIT Athena system (Mantena, 2000). Each of the campuses' connectivity is TCP/IP.

Because the central repository for code was served off a UNIX server, access from other locations and platforms was fairly easy. Thus, the ieCollab CM system falls under the category of having a CM tool and NFS. The CVS tool worked with the UNIX server to provide these two things. The only configuring by the CM team that was needed was giving access permissions to remote developers to the cee-ta.mit.edu server and providing CVS accounts to the appropriate users. If the repository had resided on a Windows 95/98/NT server as part of a LAN at MIT, however, more work would have been needed to provide access to remote users. This is because it is quite easy to access a UNIX environment from a WindowsX platform using telnet, but harder to access other kinds of servers. For example, as mentioned earlier, some sort of NFS system would have been necessary to allow users to find their ways to the files in this situation.

The CM non-code CIs were also easily accessible from the Internet. This connection required no work on the part of the CM team since all schools were connected to the Internet.

5.2.4 ieCollab CM Challenges

The CM team faced many of the challenges that any software development in a distributed and collaborative environment might face, including managerial, process-oriented, and technical challenges.

5.2.4.1 Managerial challenges

The CM team tried to work in such a way that all members contributed to everything produced by the team and every member participated in performing the CM activities. This technique was only partially successful. Involving every team member in every activity was often inefficient and even impossible for several reasons. One main reason for the difficulty was the managerial and logistical problem of having *every* developer involved in *every* activity. The three-hour time difference between Mexico and the United States was rather difficult to work around. Thus, even meeting through ICQ was difficult to coordinate. In addition, many of the students from Mexico were not at school on the weekends and did not have access to email during that time. Thus, it was difficult to communicate over these times.

5.2.4.2 Process-Oriented Challenges

Although process-oriented challenges are often of significant problem in distributed and collaborative projects, ieCollab experienced few of such problems. For example, one factor that adds many complexities to the CM process is the project size, as mentioned in Chapter 3 of this thesis. Fortunately, the ieCollab project was a very small and research-based software project. Thus, there were less lines of code to handle, for example, and many of the challenges that come with a large project were avoided in ieCollab.

Using an automated tool for controlling the access and versioning of the source code CIs was also very helpful in addressing the challenges of manual overhead in providing project control. The challenge of deciding how much control to take over the process was also not too difficult, based on the small size of the project and the freedom that was needed for development.

5.2.4.3 Technical Challenges

The technical differences between universities also affected the amount of work that could be done together. Many times, there were also additional days during which our CM colleague from Mexico was not able to check his e-mail, because of conflicts or network problems. These problems affected not only CM team member communications, but they also affected the CM team's communication with other members of the ieCollab team.

Furthermore, many ieCollab team members from Mexico also could not access the collaborate.mit.edu site, which was an important resource for documents. Thus, the unreliability of the network to Mexico is also an important challenge to note.

With respect to the general challenge related to distributed environments relating to file sharing and access, when the networks were working, ieCollab did not face an overwhelming amount of these challenges. Because each of the three universities are tolerably well connected to WANs and the Internet, it was relatively painless for CICESE and PUC to connect with the CM repository at MIT, provided that the network was functional. MIT's rich UNIX Athena system also made it particularly easy for the other schools to access the repository, since all that was necessary was telnet capabilities to the cee-ta.mit.edu server and it was not necessary to worry about interoperability.

5.3 ieCollab Configuration Management Activities

The following section describes the CM activities for ieCollab. These activities are the standard CM activities for any software development project: Configuration Identification, Configuration Control, Configuration Status Accounting, and Configuration Audits.

5.3.1 Configuration Identification

The configuration items for ieCollab were defined and identified by the CM team in the CM Plan following the guidelines provided by IEEE standards and the literature (IEEE, 1990). Table 5-3 displays the list of CIs identified for the ieCollab project, taken from the CM Plan. These CIs were named by the CM team according to the naming convention adopted in the CM Plan, which is included in the Appendix. Most of these items were available to developers at the CM/KM Web Repository.

For the completed part of the project, about eighty source code CIs were created, stored on the server, and tracked by CVS. Table 5-4 displays the list of code CIs stored in the repository for the CollabClient class to provide an example. These items were named by the programming team and versioned automatically by the CVS tool.

Because of the small size of the ieCollab project, this task of CI identification was not that difficult. CI identification was kept simple also because most of the CIs were created or originated at MIT, the university where most of the CM team was located and the repository was located. Thus, it was not as difficult to identify the CIs as if many of the CIs were created at the other universities.

Table 5-3 ieCollab Configuration Items (Liu et al, 2000)

Configuration Item Description	Configuration Item Identifier	Author	Present status
ieCollab Transaction Management Source Code	ieCollab-PR-Code-TM-*.java	PR	Available
ieCollab Meeting Management Source Code	ieCollab-PR-Code-MM-*.java	PR	Available Available
CAIRO Source Code	ieCollab-PR-CAIROCode-*.java	PR	Available
Programming Team Presentation	ieCollab-PR-Presentation.ppt	PR	Available
RA Meeting Management Specifications V 1.4	ieCollab-RA-Spec-MM-1.4.doc	RA	Available
RA Transaction Management Specifications V1.6	ieCollab-RA-Spec-TM-1.6.doc	RA	Available
RA Presentation	ieCollab-RA-Presentation.ppt	RA	Available
DE Transaction Management Specifications V 1.0	ieCollab-DE-Spec-TM-1.0.doc	DE	Available
DE Meeting Management Specifications V 0.2	ieCollab-DE-Spec-MM-0.2.doc	DE	Available
DE Client Interface Specifications V 0.3	ieCollab-DE-Spec-CI-0.3.doc	DE	Available
DE Presentation	ieCollab-DE-Presentation.ppt	DE	Available
Testing Meeting Management Specification V 2.0	icCollab-TE-Spec-MM-2.0.doc	TE	Available
Testing Transaction Management Specification V 2.0	ieCollab-TE-Spec-TM-2.0.doc	TE	Available
Testing System and Integration Specification V 1.0	ieCollab-TE-Spec-SI-1.0.doc	TE	Available
Testing Reports	ieCollab-TE-Report1.doc	TE	Available
TE Presentation	ieCollab-TE-Presentation.ppt	TE	Available
QA Plan V 2.0	ieCollab-QA-Plan-2.0.doc	QA	Available
QA Presentation	ieCollab-QA-Presentation.ppt	QA	Available
CM Plan V. 1.0	ieCollab-CM-Plan-1.0.doc	CM	Available
CM Presentation	ieCollab-CM-Presentation.ppt	CM	Available
KM Plan	ieCollab-KM-Plan.doc	KM	Available
KM Presentation	ieCollab-KM-Presentation.ppt	KM	Available
User Manual	ieCollab-U_manual.doc	KM	Available

Table 5-4 Sample CIs for ieCollab from the CollabServer class

CI Description	CI Name	Revision Number	Author
GUI code	Create.java	1.2	bharath
GUI code	CreateMeeting.java	1.2	bharath
GUI code	CreateWorkgroup.java	1.2	bharath
GUI code	EditMeeting.java	1.2	bharath
GUI code	EditWorkGroup.java	1.2	bharath
GUI code	ErrorMessage.java	1.2	bharath
GUI code	HelpWindow.java	1.1	bharath
GUI code	IECollab.java	1.2	bharath
GUI code	LoginWindow.java	1.2	bharath
GUI code	ProfileWindow.java	1.2	bharath
GUI code	RegistrationWindow.java	1.2	bharath
GUI code	SearchWindow.java	1.2	bharath
GUI code	UserPubInfoWindow.java	1.2	bharath

5.3.2 Configuration Control

As with many distributed projects and as mentioned earlier, the ieCollab project utilized an automated tool to aid in configuration control, CVS. Using a CM tool made configuration control much easier. To recap the function of CVS, CVS enabled developers to have remote access to the project repository by copying the requested file onto the user's local directory. Thus, any authorized user could access and modify a CI this way and update the repository with changes when ready up until the CI was approved and was added to the baseline. When updating the repository, the user created a log of what he/she had changed, and CVS automatically assigned a version number to the changed code. The user was also able to look at older versions of the code or compare various versions for differences.

This method of file version control provided freedom and visibility to the developers while supplying a necessary degree of control. It also encouraged more collaboration since everyone could look at code and work together. This method also made it easy to review changes made by others and even make further changes, which is precisely what occurred in the ieCollab project. An important thing to note, however, is that once a CI was baselined, the CI was considered "locked." This CI was then unavailable for change without an approved of a change request.

Though version control was quite extensive with CVS, the change control process was not included in this automation, primarily because CVS does not provide change capabilities. However, as with many distributed development projects, the change control process for ieCollab was quite controlled. This control was achieved through a set change control protocol, which is illustrated in Figure 5-2.

Following standard procedures found in the IEEE standards (IEEE, 1990) and suggestions from the literature, the key component of the ieCollab change control process was the change request. This change request was necessary before any changes to baselined CIs were considered. A sample ieCollab Change Request Form can be viewed in the Appendix as part of the ieCollab CM Plan. This form requests information such as a description of the change, reason for change, priority of change, CIs affected by change, and other information. Once this request was approved as a submission from the entire team, it was reviewed by the CCB. The CCB then decided to either approve or deny the change request and notified the appropriate people that this decision affected. If the CCB approved the request, the appropriate people were then "allowed" to check the CI out of the project repository for changes. Once the changes were made by the appropriate people, they were then checked into the repository, where a new baseline was established for testing and quality check. If all the changes were done according to CM standards and procedures, this change would result in a new version of the baselined CI.

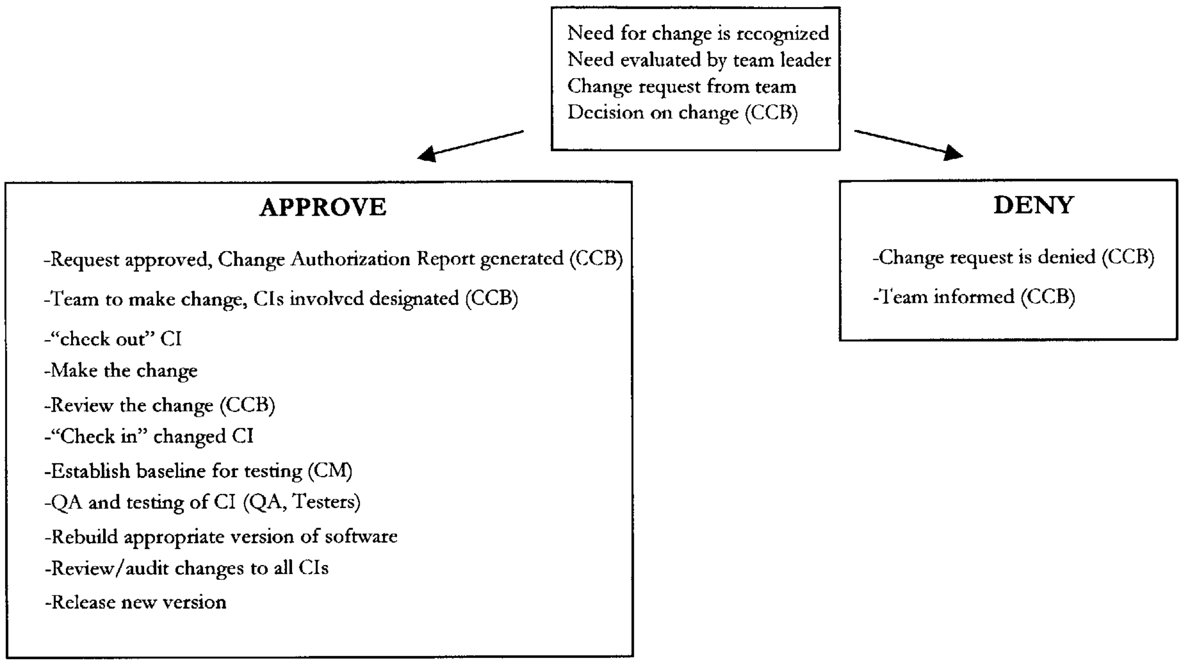

Need for change is recognized
Need evaluated by team leader
Change request from team
Decision on change (CCB)
APPROVE
-Request approved, Change Authorization Report generated (CCB)
-Team to make change, CIs involved designated (CCB)
-"check out" CI
-Make the change
-Review the change (CCB)
-"Check in" changed CI
-Establish baseline for testing (CM)
-QA and testing of CI (QA, Testers)
-Rebuild appropriate version of software
-Review/audit changes to all CIs
-Release new version
DENY
-Change request is denied (CCB)
-Team informed (CCB)

Figure 5-2 ieCollab Change Control Protocol (Liu and Mantena, 2000)

A similar process applied to non-code items. However, a change request to a non-code item often did not always involve physically changing the document, but often it involved changing a concept in the document. In this situation, after the CCB approved the change, the developer would produce a document describing the change in concept, and this document would be linked to the affected document.

It is evident through this discussion that the entire change process required quite a bit of manual overhead. For instance, it required spending time to download the change request form from the CM/KM Web Repository, filling it out either by hand or by computer, and then e-mailing it to CM or posting it to the web, all of which required quite a bit of effort. Though this entire process was never fully put into action in the ieCollab project because of time constraints, had it been utilized, it would have been somewhat tedious. After requesting the change, the requestor would have to wait for the change request to reach the proper people and then wait for the CCB to meet and decide how to respond to the request. Add to these things trying to organize all-member CCB meetings to collaboratively approve change requests and the process became extremely long and drawn out.

It is also apparent from this process that change control was very tightly controlled in the ieCollab project. Using the described method, frequent changes to baselined CIs were discouraged, since approved CIs were "locked" and could only be changed after having a formal change request approved.

5.3.3 Configuration Status Accounting and Configuration Audits

As predicted in Chapter 4, configuration status accounting and configuration status audit practices for ieCollab did not change fundamentally from the standard practices for non-distributed CM. The automation of configuration control did in general mean that less work was needed to carry out configuration status accounting and configuration audits, however. For example, the ability to view the differences in code between versions using CVS was very useful for reporting what changes were currently being made or whether a change had been implemented. In the same way, viewing differences also provided an audit trail for QA and CM to follow when making sure that the changes on a CI were carried out properly.

5.4 CM Tools

As already discussed, incorporating tools into the CM process have had an enormous impact on distributed collaboration in software development, and these tools were a necessity in the ieCollab project as well. The main CM tools used for this project were CVS and the CM/KM Web Repository.

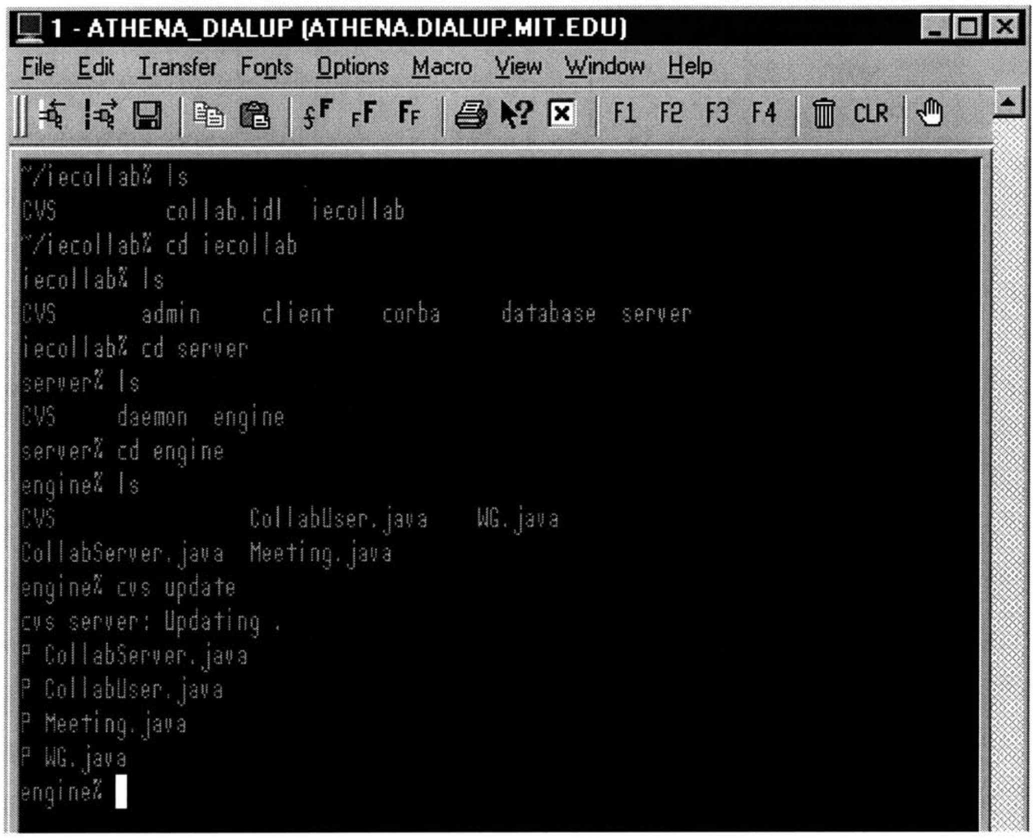
5.4.1 CVS

CVS (Concurrent Versions System) is a popular CM tool, used by many distributed development projects. Similar in many ways to RCS, it is a version control system, which enables a project to keep a history of source files (Whitehead, 1999).

CVS is particularly useful in a distributed environment because it allows users to access the project repository from remote locations. It is also very useful in a collaborative environment. As already asserted, in a collaborative environment where there are many people working together, it is very easy for disasters such as overwriting commonly used code. CVS helps by allowing developers to work on the files they need while being insulated from other developers. The changes made are merged with the changes of other developers when each developer is done (Cederquist et al, 1993) using the update-edit-commit work cycle (Whitehead, 1999). The way that CVS saves each version is actually by saving the *changes* that were made, not an entire copy of each version. This helps to save space in the database (Kliewer, 1998).

Although CVS does make configuration control much easier, it is not a substitute for management, as there are many tasks that still should be carefully monitored, such as schedules and releases (Cederquist et al, 1993). CVS also does not have change control. Although certain functionalities of CVS can be used to aid in change control, such as the "diff" function, they must be used in conjunction with management so that nothing falls through the cracks.

CVS worked fairly well for the managing the ieCollab project. It provided a good amount of control for versioning and allowing users to access CIs. However, one challenge with CVS is the merging of changes from many developers on the same CI. Even though these merges are automatic, it is not advisable to leave CVS to do this without monitoring the merges to make sure they were done properly. CVS does recognize anomalies and will notify users of these merges; however, it is difficult to tell whether the finished product will represent what was really meant by developers. This problem was not experienced in the ieCollab project; however, it might have been more of a concern with more lines of code and more developers working on the same modules.

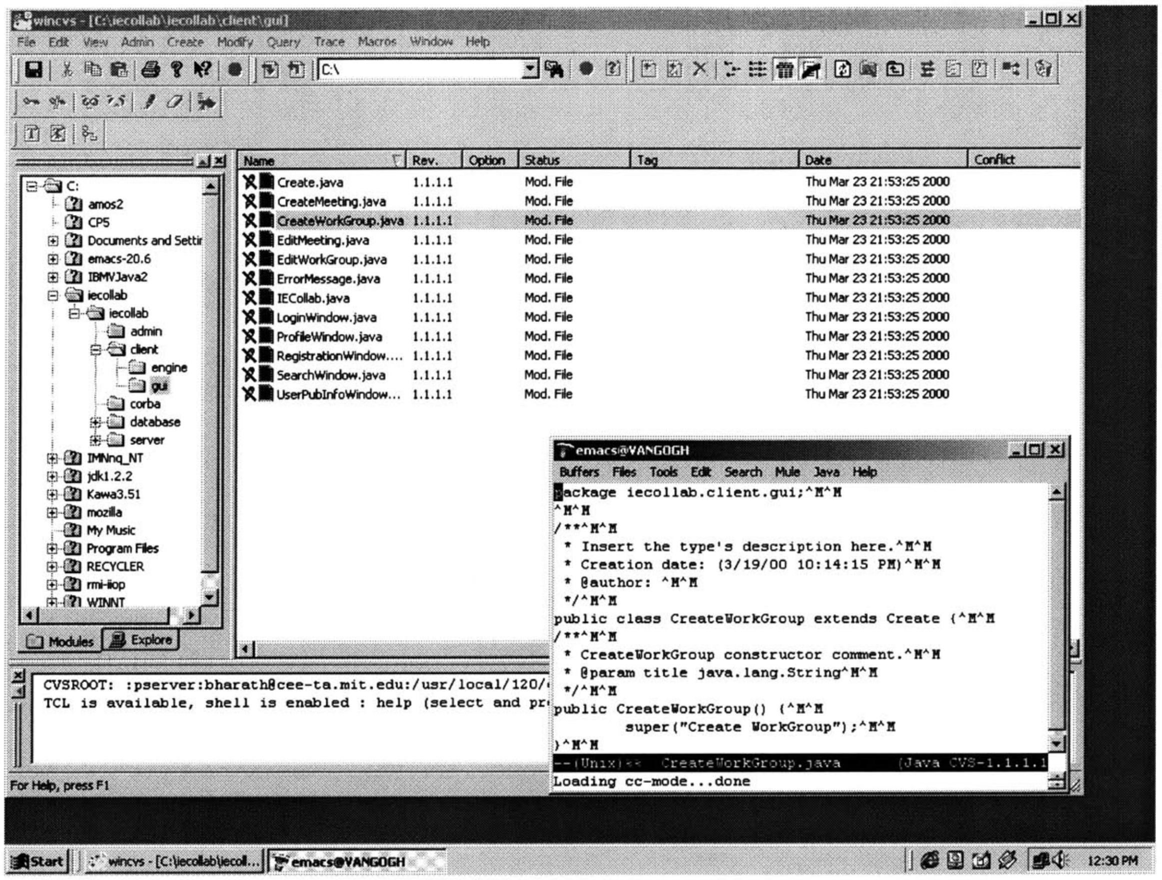

1 - ATHENA_DIALUP (ATHENA.DIALUP.MIT.EDU)
File Edit Transfer Fonts Options Macro View Window Help
F1 F2 F3 F4 CLR

```
~/iecollab% ls
CVS        collab.idl  iecollab
~/iecollab% cd iecollab
iecollab% ls
CVS      admin     client     corba     database  server
iecollab% cd server
server% ls
CVS     daemon  engine
server% cd engine
engine% ls
CVS                 CollabUser.java    WG.java
CollabServer.java  Meeting.java
engine% cvs update
cvs server: Updating .
P CollabServer.java
P CollabUser.java
P Meeting.java
P WG.java
engine%
```

Figure 5-3 CVS UNIX interface

Aside from the traditional UNIX view for CVS, which is shown in Figure 5-3, the ieCollab project also made use of WinCVS. Used by only a few developers, WinCVS is a Windows user interface for CVS on the front end, supported on the back end by the cee-ta.mit.edu server. The WinCVS interface is shown in Figure 5-4. This application allowed users to perform the same CVS commands by using a graphical user interface.

Another alternative to accessing CIs through the UNIX front-end was CVS Web interface, implemented toward the middle of the project's programming process. The CVS Web interface allowed users to view CVS tree of CIs on the Web but did not allow users to upload or change these CIs. However, the Web interface did allow users to view



Name	Rev.	Option	Status	Tag	Date	Conflict
Create.java	1.1.1.1		Mod. File		Thu Mar 23 21:53:25 2000	
CreateMeeting.java	1.1.1.1		Mod. File		Thu Mar 23 21:53:25 2000	
CreateWorkGroup.java	1.1.1.1		Mod. File		Thu Mar 23 21:53:25 2000	
EditMeeting.java	1.1.1.1		Mod. File		Thu Mar 23 21:53:25 2000	
EditWorkGroup.java	1.1.1.1		Mod. File		Thu Mar 23 21:53:25 2000	
ErrorMessage.java	1.1.1.1		Mod. File		Thu Mar 23 21:53:25 2000	
IECollab.java	1.1.1.1		Mod. File		Thu Mar 23 21:53:25 2000	
LoginWindow.java	1.1.1.1		Mod. File		Thu Mar 23 21:53:25 2000	
ProfileWindow.java	1.1.1.1		Mod. File		Thu Mar 23 21:53:25 2000	
RegistrationWindow....	1.1.1.1		Mod. File		Thu Mar 23 21:53:25 2000	
SearchWindow.java	1.1.1.1		Mod. File		Thu Mar 23 21:53:25 2000	
UserPubInfoWindow...	1.1.1.1		Mod. File		Thu Mar 23 21:53:25 2000	

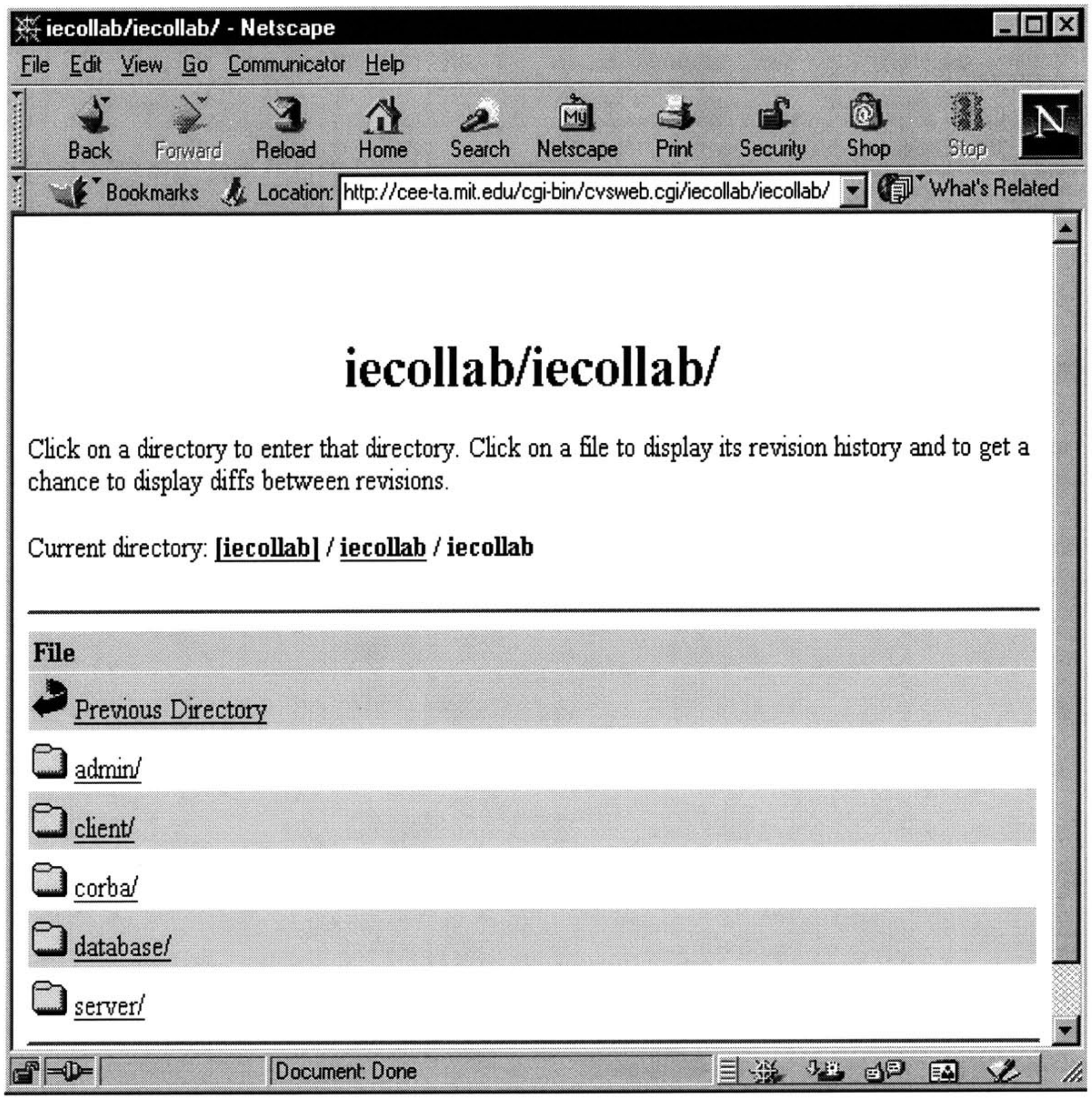
```
package iecollab.client.gui;^M^M
^M^M
/**^M^M
 * Insert the type's description here.^M^M
 * Creation date: (3/19/00 10:14:15 PM)^M^M
 * @author: ^M^M
 */^M^M
public class CreateWorkGroup extends Create {^M^M
/**^M^M
 * CreateWorkGroup constructor comment.^M^M
 * @param title java.lang.String^M^M
 */^M^M
public CreateWorkGroup() {^M^M
        super("Create WorkGroup");^M^M
}^M^M
--(Unix)**  CreateWorkGroup.java       (Java CVS-1.1.1.1
Loading cc-mode...done
```

```
CVSROOT: :pserver:bharath@cee-ta.mit.edu:/usr/local/120/
TCL is available, shell is enabled : help (select and pr
```

Figure 5-5 WinCVS ieCollab Project interface

the differences between versions of a CI. Figure 5-5 displays the CVS Web interface. Most developers found this tool extremely useful and easier to use than the traditional UNIX method, although the UNIX method was still the most popular method of accessing CIs, since the UNIX interface was not limited in any way to the functions that could be performed.

One automated feature that would have been useful for a CM tool for ieCollab is change control. It would have been useful to use a CM tool that also automated the change request process. For example, PVCS (Process Version Control System) is one such product that also acts as a change manager.


iecollab/iecollab/ - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location: http://cee-ta.mit.edu/cgi-bin/cvsweb.cgi/iecollab/iecollab/
What's Related
Document: Done

iecollab/iecollab/

Click on a directory to enter that directory. Click on a file to display its revision history and to get a chance to display diffs between revisions.

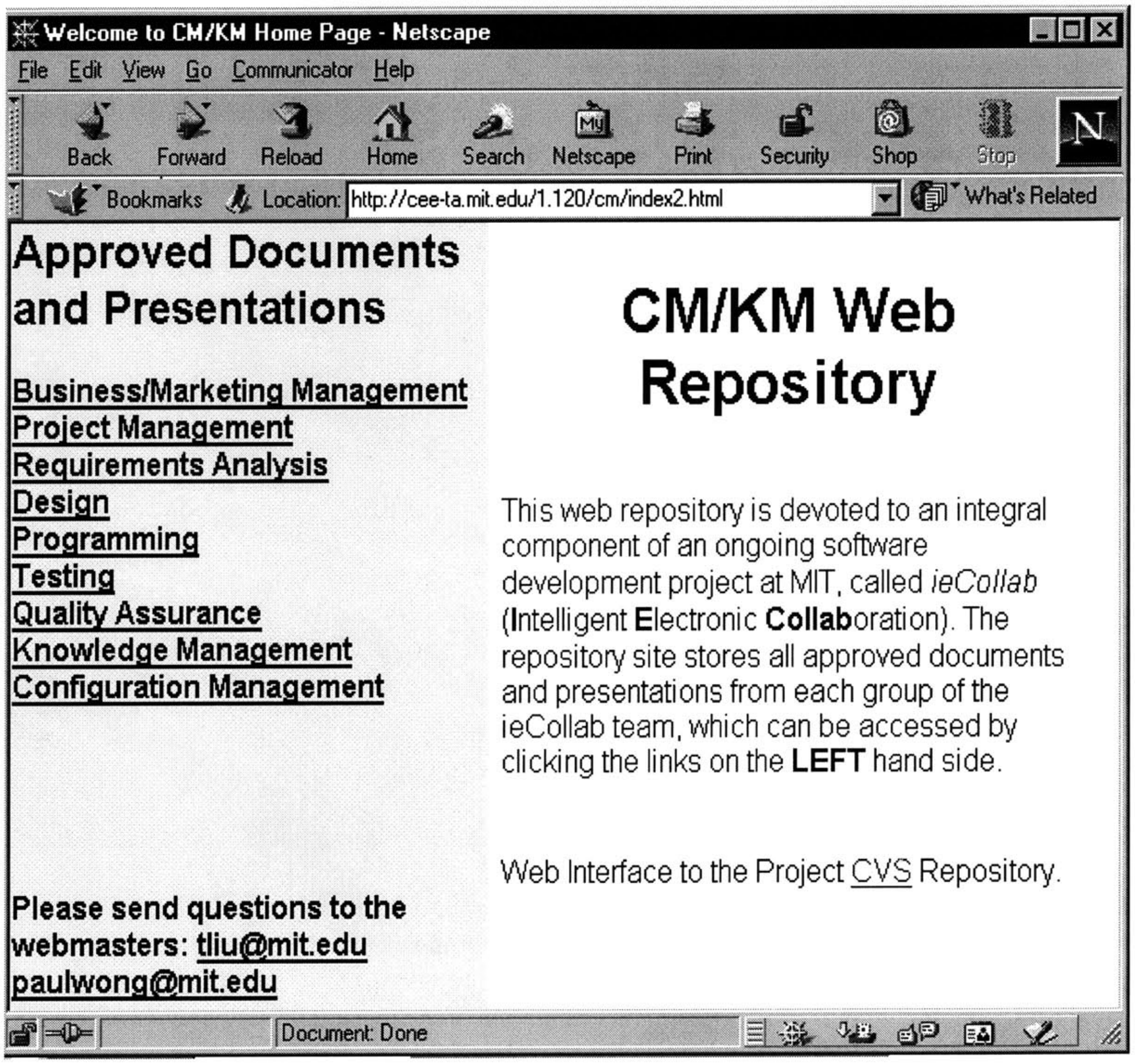
Current directory: [iecollab] / iecollab / iecollab

File
Previous Directory
admin/
client/
corba/
database/
server/

Figure 5-5 CVS Web Interface

5.4.2 CM/KM Web Repository

As mentioned earlier, non-code configuration items also resided on the cee-ta.mit.edu server; however they were not under the control of CVS but rather the CM/KM Web Repository. This web site was located at URL http://cee-ta.mit.edu/1.120/cm/index2.html.


Welcome to CM/KM Home Page - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location: http://cee-ta.mit.edu/1.120/cm/index2.html
What's Related
Document: Done

Approved Documents and Presentations

Business/Marketing Management
Project Management
Requirements Analysis
Design
Programming
Testing
Quality Assurance
Knowledge Management
Configuration Management

Please send questions to the webmasters: tliu@mit.edu paulwong@mit.edu

CM/KM Web Repository

This web repository is devoted to an integral component of an ongoing software development project at MIT, called *ieCollab* (**I**ntelligent **E**lectronic **Collab**oration). The repository site stores all approved documents and presentations from each group of the ieCollab team, which can be accessed by clicking the links on the **LEFT** hand side.

Web Interface to the Project CVS Repository.

Figure 5-5 CM/KM Web Repository (Whitehead, 1999).

Unlike KM's collaborate.mit.edu web repository, the CM/KM Web was meant to be more of a static "snapshot" of documents that were either approved or nearing approval, the final authority on project non-code CIs, as defined by the project advisor. Another of the site's purposes was to provide easy viewing and copying opportunities of documents for a Windows environment. Most of the documents in this repository were in either .doc or .ppt format. Thus, if these CIs had been managed by CVS, they would have needed to be downloaded and then ftped to a Windows environment for viewing since the CVS repository runs only on a UNIX environment.

Having this site on the Web was useful since all of the collaborating universities either primarily use or have access to Windows applications and using a web browser with a plug-in would enable immediate viewing of the document by the correct Windows application. This web site was accessible to any developer with Internet connection and a web browser, and documents were also available for download. However, uploading capabilities were not available to the general user as they were for the KM collaborate.mit.edu project site. Only CM team members were given upload permissions through the cee-ta.mit.edu server. Figure 5-5 displays a sample page from the CM/KM Web repository.

5.5 Problems and Recommendations for ieCollab Configuration Management

Throughout the project, the CM team ran into a number of difficulties and problems. Some of these difficulties could have easily been avoided and others were unavoidable. Thus there are different things that could be tried to solve these problems or at least alleviate the pain in the future. Generally, relating to the ieCollab CM team, the problems that were encountered can be described using the following categories: knowledge, collaboration, and technology problems.

5.5.1 Knowledge Problems

One of the most obvious problems that the CM team faced was only realized toward the end of the project. This problem is that CM had not been practiced as a continual process throughout the duration of the project since the CM team (and project management) was under the impression that CM was practiced only at the end of the project during code generation. The CM team did not realize that this was a problem until researching the role of CM late into the project. As the programming period approached, it became clear that CM would be absolutely necessary. Essentially, the CM team was not knowledgeable about its role. The consequences of this mistake were not grave, considering the limited timing of the project and the many delays experienced. However, it did mean that the configuration management team did not do much meaningful work until very late into the project. Certain tasks that ought to have been done early on in the project, such as the identification of CIs, were also not done until later, and standards that should have been set by CM early on, such as naming standards, were difficult to enforce at the end of the project. Because the CM team was unaware of what it needed to do from the beginning of the project, it was unable to prepare key CM tools ahead of time. For example, it would have been very useful to set the CVS repository and CM Web Repository up before programming began. Although the CM team was able to set these tools up in time, they would have been much better organized with advance preparation. The CM team was also unclear about its tasks with respect to the knowledge management team, which was a team not typical to the software development process and rather unique to the ieCollab project.

One way that CM team knowledge could have been improved is through the provision of CM guidelines, outlining key tasks that should be researched from the beginning of the project. Although it would have been possible for the CM team to research this information themselves, there was little or no incentive. Furthermore, the CM team was in a way misled by the project management team, who did not really understand CM well either. The PM team set a very late date for the submission of the CM Plan. Because this date was set to be so late, this created the impression that CM processes did not need to occur until after the due date. The creation of the CM Plan would have been a necessary task to have completed in order for CM to begin its work.

5.5.2 Collaboration Problems

Although communication lines between CICESE and MIT CM team members were relatively open, collaboration with the team member in CICESE was still difficult at times, from MIT's point of view. For example, the only CICESE team member, although he was sufficiently active during the beginning of the project, "disappeared" from the CM team toward the end of the project, not responding to e-mails about contributing to documents, presentations, and other tasks.

Although the disappearance of that team member was never explained, other communications deficiencies between different universities can be explained by a number of reasons. The first reason is the lack of accountability from not having regular face to face communication. As the forms of communication most commonly used in the project such as e-mail and ICQ are fairly anonymous, it was very easy for team members to feel less pressure to accomplish tasks. Furthermore, as it was often in the best interest of time and efficiency to divide tasks separately between universities, team members most likely became very comfortable working only with students from the same location, leaving little incentive for collaboration.

Technological difficulties also very likely affected the collaboration problems experienced in the CM team and in the ieCollab team in general. For example, though the plan had been to broadcast all team presentations that occurred at MIT to CICESE, because of difficulties with the video conferencing, this never occurred. Thus, it is very possible that the CICESE campus did not feel as much a part of the ieCollab team. Furthermore, it was very common for servers at Mexico to be down so that project web sites could not be accessed and even e-mail could not be received. This served to hinder collaboration as well in CM team communications. Another example of the limitations of technology's affecting collaboration occurred because of the difficulty of hosting a mirror site to the CVS repository at CICESE. Since this was a key component of the CM process, from a collaborative point of view, it would have been very beneficial to mirror this site at Mexico so that the CM team at CICESE could also take part in maintaining part of the repository. However, in examining this possibility, it became apparent that this would not be possible because of technology limitations. Mexico essentially decided that mirroring the site would also not be worth the time and effort. Thus, all of the CVS administration and monitoring occurred at MIT without any collaboration from the CM team in CICESE.

It is difficult to say how these collaborative problems described above could have been solved. One way perhaps to address the accountability problem would be to have a meeting before the start of the project so that all CM team members and ieCollab members could meet and develop a sense of camaraderie and mutual respect. Another technique that might have worked would be to enforce a weekly meeting between distributed team members to touch base, using the team's technology of choice, where the default would be long distance phone. Enforcing this meeting would make it so that there is no way that team members can ignore or put off the tasks that need to be done. Even if a task could not be done, this communication would at least alert other team members of potential problems.

In addressing the technological problems that influence CM problems, it has been very difficult for those at MIT to provide a solution for how to overcome network problems with CICESE or PUC when almost all problems with technology so far have been from the side of Mexico or Chile. Thus, perhaps the best way to deal with these problems would be not to try to "fix" the technology, but to ask teams to find alternative ways to communicate, such as by telephone.

5.5.3 Miscellaneous

Many of the problems that occurred were out of the CM team's control. For example, after the CM team set up the CVS repository for the first time and put a great deal of time and resources into it, the repository was completely wiped off the server in an accident. This resulted in CM team having to reinstall the CVS program on a new server, which was very time consuming and frustrating. A solution to this problem might have been to provide a dedicated server for the CM tasks. This would have ensured that only the CM would be able to access this server. It is a feasible suggestion, since in most cases, the CM repository must hold many files, be freely accessible, and provide security measures, and requirements are not always possible to fulfill when the server is shared by other applications.

5.6 Summary

Thus, to summarize the main points of this chapter, one of the main goals of the ieCollab project was to demonstrate and study collaboration in distributed software development. This goal was met by allowing students to practice software development roles first hand in a distributed and collaborative environment.

ieCollab's CM process is a good example of how the configuration management role is carried out in a real distributed and collaborative environment. By studying the CM process and experiencing it first hand, it is possible to understand the challenges of distributed and collaborative CM and evaluate what can be done to address these challenges. Many times, technology and tools are both a help and a hindrance.

For example, the distributed and collaborative environment makes some kind of CM tool an almost must. These tools free up people's time by automating certain processes, and this time can now be spend focusing on more meaningful things, such as improving collaboration. However, the technology needed for this kind of collaboration also often proves to be of great trouble, as experienced in the ieCollab project. Nonetheless, learning from the problems and successes that this project experienced will help determine what kind of developments would be useful for CM in the future.

6. Future Developments for Distributed and Collaborative Configuration Management

Distributed and collaborative CM is a changing process. Distributed networks have only started to take off within the last five to ten years, and as technology is always rapidly changing, it is clear that distributed networks have yet to reach maturity. Furthermore, the opportunities for collaboration are still just being discovered. The organization of the future will be the "virtual organization," a development team created from different organizations collaborating together on a specific project across a wide area network and then disbanding at the end to form new alliances and virtual organizations. These teams will want to maintain autonomy of their own processes, policies, tools, and environments; however there will still need to be set CM policies for the project, data sharing, and common items to the project. The questions that remain to be answered then is--how can a project adopt and enforce a common policy for CM while preserving the autonomy of the individual organizations? (Noll and Scacchi, 1997) Many aspects of CM will have to be examined to answer this question, such as the CM system architecture.

6.1 System Architecture

The architecture of CM systems is continuing to evolve as a result of the changing nature of distributed and collaborative CM. Components especially affected are the CM repository and network.

6.1.1 CM Repository

In the future of configuration management, the architecture of the CM repository will most likely need to change. As projects grow larger, more distributed, and more collaborative, the currently popular global centralized repository may turn out to be inadequate to handle these growing needs. The centralized repository where all CIs are stored on one server works very well for many organizations; however, it has many disadvantages, such as the following (Allen et al, 1995):

- Subjecting this central repository to frequent accesses will make it susceptible to network problems such as partitioning.
- Performance bottlenecks at peak loads.
- Central repository suffers from lack of scalability. As the system grows larger, the repository will probably not be able to handle the larger loads of users since it is located on a single server.

One possible solution to this problem is having a repository that is decentralized physically, but centralized logically. In this situation, the CIs are located on different nodes, along the network, but are viewed as one repository. This configuration is perfect for a virtual organization, since each individual organization that joins the project will have its own system, and this configuration will allow the repositories to be joined logically. An example of a tool that can handle this kind of task is NUCM (Network-Unified Configuration Management), a testbed being developed that provides a model for a distributed repository and common interface and allowing the implementation of two very different CM policies (van der Hoek et al, 1996). Figure 6-1 illustrates the infrastructure for wide area software development for a similar project done by AT&T called GRADIENT (Global Research and Development Environment). GRADIENT takes multiple repositories (one in Taiwan and one in Murray Hill, NJ, for example) and allows users to access files on all repositories, while maintaining the autonomy of each individual site.

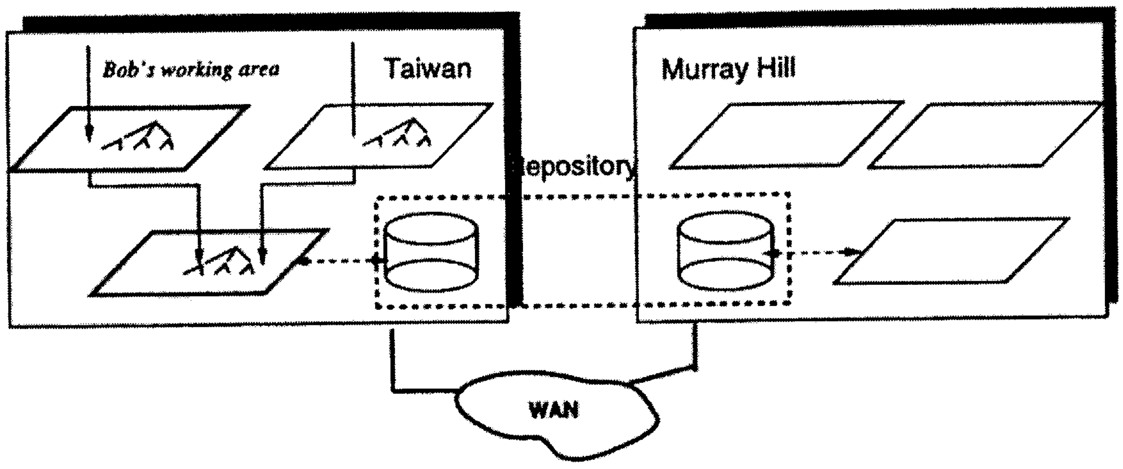


Figure 6-1 Multiple Repositories Accessed as One across a WAN (Belanger et al, 1996)

6.1.2 CM Network

Another feature that will be almost mandatory for truly distributed and collaborative CM is interoperable CM systems. Because individual organizations will have their own CM systems in place, it will be necessary to be able to combine these systems under one system. Some CM systems on the market already have some interoperability with other systems. For example, ADC allows the importation of SCCS files into the ADC repository (Dart, 1992).

It is generally acknowledged that "in most software projects, collaborating sites are loosely connected and poorly integrated" (Belanger, 1995). Thus, CM networks of the future will require more tightly coupled and integrated systems.

6.2 CM Process

The CM process is also changing to reflect the changing software development environment. Many of these changes have to do with the amount of control that is exercised over the process. For example, more and more, people are favoring more loosely controlled change control. Traditional CM is rather tightly controlled, and frequent changes are discouraged, since they create manual overhead and delays. Organizations may be tempted to have exert more control as projects become larger, and tend to get a bit out of control. However, for the future, the exact opposite may be needed.

According to the literature, though control is very important to managing a software development project, it is also important that an organization balance the amount of CM control it uses carefully. In a distributed environment particularly, too much control becomes counterproductive, choking the project activities with paperwork and adding to the difficulties of being distributed. It should be mentioned, however, that a certain amount of control is necessary to track the work done by developers and prevent errors

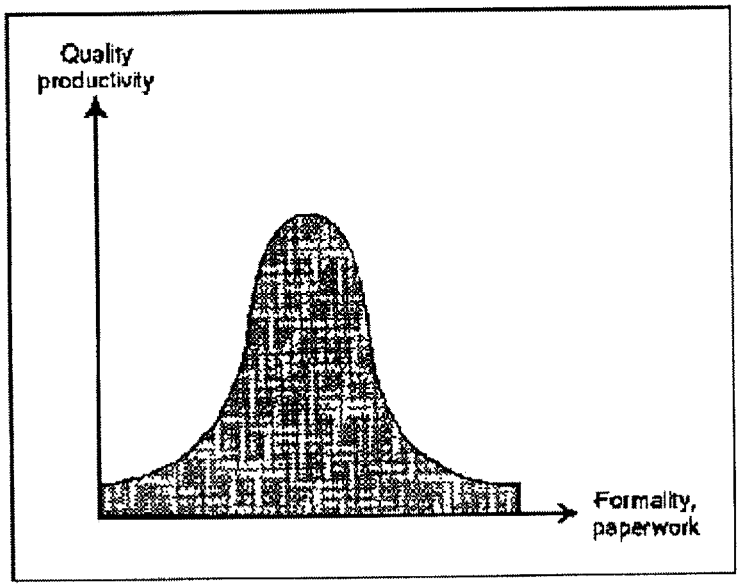


Figure 6-2 Quality of CM Process as a Function of Formality (Kliewer, 1998)

and mistakes. As shown in Figure 6-2, as formality and paperwork in CM (i.e. control) increase toward the optimal level, quality and productivity also increase. However, once this control reaches too great an amount, the quality and productivity of a project decrease (Kliewer, 1998).

6.3 CM Tools

CM tools will continue to play an integral part in configuration management in the future. Along with versioning systems, web CM systems are now available to enable more collaboration than ever before.

6.3.1 Versioning Systems

Although many versioning systems have been discussed in this thesis, many other exist and provide very good functionality. However, the nature of software development is such that every project is different and has different needs in a CM tool. Other tools that are important consider include ClearCase, DCVS (Distributed CVS), and Continuus/CM, among others.

6.3.2 CM on the Web

One technology that is really changing the way that projects and teams interact is the Internet and the World Wide Web. Because of the ease of access and its near-ubiquitous state, it is an excellent medium for distributed collaboration. And now, according to the literature, "web-based development is a reality."

For example, there are currently some research efforts under way to provide versioning and configuration management capabilities for the Web. One such product is "Delta-V." Delta-V aims to allow editing of source code, documents, and other CIs, all on the web. Documents are edited directly onto a web server instead of a download. Although this concept is still very new, it is currently being tested in applications such as Internet Explorer 5 (Whitehead, 1999).

However, utilizing the web for CM would introduce security issues. The openness of the Internet and Web based languages such as HTML and Java are ideal for developers, but also ideal for intruders. Encryption methods would be needed to ensure proper security (Gumaste et al, 1996).

6.4 Conclusion

In conclusion, configuration management is a vital part of any software development project. As development environments become more distributed and software development becomes more collaborative, good CM will be of even greater importance than before, adding a measure of necessary control to a rather chaotic situation. By ensuring the integrity of software configuration items over the evolution of the product, CM helps to make a project more stable and repeatable, as well.

With this relatively new distributed and collaborative environment comes many challenges to CM. For example, how will the CM team itself deal with these changes, and how will the rest of the development team adjust to the changes with respect to the CM process? Furthermore, there are many factors to consider when planning out a proper CM protocol for distributed collaboration, such as the technology needed to make it successful, managerial and logistical issues, and core process issues that may need to change.

ieCollab is a good example of a distributed software development project where collaboration was particularly emphasized. In this project, it was possible to examine configuration management in a distributed and collaborative environment and apply certain technologies and protocols to experiment on what makes CM work in this kind of environment. Although it is difficult to give definitive answers to these questions, much was learned in during the project, and what was particularly important was the learning that can be applied to future developments for CM.

References

Allen, Larry, Gary Fernandez, Kenneth Kane, David Leblang, Debra Minard, and John Posner. "ClearCase MultiSite: Supporting Geographically-Distributed Software Development." in *Software Configuration Management: ISCE SCM-4 and SCM-5 Workshops Selected Papers.* Springer: New York, 1995.

Abbott, Erik, Joao Arantes, Ivan Limansky. "Project Management Plan Version 1.2." for ieCollab. MIT, 2000.

Belanger, Dave, David Korn, and Herman Rao. "Infrastructure for Wide-Area Software Development." inn *Software Configuration Management: ISCE'96 SCM-6 Workshop Selected Papers.* Springer: New York, 1996.

Ben-Menachem, Mordechai. *Software Configuration Management Guidebook.* McGraw-Hill: London, 1994.

Bersoff, Edward and Alan Davis. "Impacts of Life Cycle Models on Software Configuration Management." *Communications of the ACM.* August, 1991. from http://www.acm.org.

Brown, Alan W., Peter H. Feiler, and Kurt C. Wallnau. "Understanding Integration in a Software Development Environment." SEI: Pittsburgh, 1992.

Buckley, Fletcher J. *Configuration Management: Hardware, Software, and Firmware.* IEEE: New York, 1993.

Cederquist, Per et al. "Version Management with CVS for CVS 1.10." Signum, 1993. from http://www.loria.fr/~molli/cvs/doc/cvs_1.html. Last visited 3/5/00.

Chan, Agnus K. F and Sheung-lun Hung. "Software Configuration Management Tools." IEEE paper. IEEE, 1997. From http://www.ieee.org.

Chen, Hao, Wassim Solh, Manuel Alba, Sugata Sen, and Anup Mantena. "ieCollab Transaction Management Design Specification 1.0." MIT, 2000.

Ci, James X., Mustafa Poonawala, and Wei-Tek Tsai. "ScmEngine: A Distributed Configuration Management Environment on X.500." in *Software Configuration Management: ISCE'97 SCM-7 Workshop Proceedings.* Springer: New York, 1997.

D'Amico, Mary Lisbeth. "Siemens Previews High-Tech Gadget." http://www.PC World.com/shared/printable_articles/0,1440,9728,00.html

Dart, Susan. "The Past, Present, and Future of Configuration Management." Technical Report. SEI:CMU, 1992. from http://www.ieee.org

Dart, Susan. "Spectrum of Functionality in Configuration Management Systems." Technical Report. SEI:CMU, 1990. from http://www.ieee.org

Favela, Jesus. "Configuration Management." Instructive presentation for 1.120. November,1998.

Feiler, Peter. "Software Process Support Through Software Configuration Management." SEI Paper. IEEE, 1990.

Gumaste, U.V., Ti-Chung R. Hsueh, Andrew A. Nocera, and Yiu Kwan Wo. "Configuration Management Strategy for Distributed and Diverse Software Development Environments." Proceedings of The IEEE International Conference on Industrial Technology. IEEE, 1996. from http://www.ieee.org.

IEEE. "Standard for Software Configuration Management Plans: IEEE Std 828-1990." IEEE, 1990.

Kliewer, Chris. "Software Configuration Management." University of Calgary. 1998. from http://www.sern.uclagary.ca/~kliewer/SENG/621/SCM_Essay.html.

Kramer, Jeff. "Distributed Software Engineering." IEEE paper. IEEE, 1994.

Liu, Teresa, Anup Mantena, and Manuel Alba. "Configuration Management Plan For ieCollab Version 1.0." MIT, 2000.

Liu, Teresa and Anup Mantena. "ieCollab CM Presentation." MIT, 2000.

MacKay, Stephen A. "State of the Art in Concurrent, Distributed Configuration Management." in *Software Configuration Management: ISCE SCM-4 and SCM-5 Workshops Selected Papers.* Springer: New York, 1995.

Mantena, Anup. Personal correspondence on April 15, 2000.

Noll, John and Walt Scacchi. "Supporting Distributed Configuration Management in Virtual Enterprise." in *Software Configuration Management: ISCE'97 SCM-7 Workshop Proceedings. Springer: New York, 1997.*

Paulk, Mark C., Bill Curtis, Mary Beth Chrissis, and Charles V. Weber. "Capability Maturity Model for Software, Version 1.1." SEI Technical Report. February, 1993.

Pressman, Roger S. *Software Engineering: A Practitioner's Approach.* McGraw-Hill: New York, 1997.

Scacchi, Walt. "Models of Software Evolution: Life Cycle and Process." SEI paper. SEI: Los Angeles, 1987. from http://www.ieee.org.

van der Hoek, Andre, Dennis Heimbigner, and Alexander L. Wolf. "A Generic Peer-to-Peer Repository for Distributed Configuration Management." Proceedings of ISCE-18. IEEE, 1996.

www.whatis.com. Last visited, May 5, 2000.

Whitehead, Jim. "The Future of Distributed Software Development on the Internet: From CVS to WebDAV to Delta-V." *Web Techniques Magazine.* Freeman, 1999. from http://www.webtechniques.com/archives/1999/10/whitehead.html.

Williamson, Mickey. *Automated Software Configuration Management: Issues, Technology, and Tools.* Cutter Information Group: Salem, 1990.

Wong, Paul Koon Po. "Knowledge Management Plan v 1.0 for ieCollab." MIT, 2000.

List of Appendices

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
CIVIL AND ENVIRONMENTAL ENGINEERING DEPARTMENT

1.120 Information Technology Master Engineering Project
Distributive Software Engineering Laboratory
Fall 1999, IAP 2000, Spring 2000

Configuration Management Plan For ieCollab Version 1.0

Updated by: Teresa M Liu, Anup V Mantena
Participants on Modification :
Manuel Alba, Anup V Mantena, Teresa M Liu

3/12/00

References and Links:
IEEE Standard for Software Configuration Management Plans 828-1990

Table of Contents

1. Introduction

This Software Configuration Management Plan describes the Configuration Management (CM) organization and practices applied consistently and uniformly throughout the life cycle for Computer Software Configuration Items that are developed or maintained by 1.120 for the ieCollab project. Software Configuration Management is the process used during software development and maintenance to identify, control, and report functional and physical configurations of software products (e.g., source code, executable code, databases, test scenarios and data, and documentation).

1.1 Purpose

The purpose of this document is to define CM responsibilities (requirements), resources, and processes used during the development and maintenance of the ieCollab system. By reading this document, all interested will be able to gain a better understanding of the role of CM's role in the ieCollab project.

1.2 Scope

The scope addresses CM applicability, limitations, and assumptions on which the Plan is based.

1.2.1 Project Overview

This plan establishes the CM methods used during the development and maintenance of the ieCollab system. ieCollab is an Internet-based collaborative application service provider for communicating information and sharing software applications in a protocol-rich Java meeting environment. It will be the result of a joint effort of students from the Massachusetts Institute of Technology in the United States, Pontificia Universidad Catolica (PUC) in Chile, and Centro de Investigacion Cientifica y de Educacion Superior de Ensenad (CICESE) in Mexico as a project for the class "Distributed-Collaborative IT Development," also known as 1.120.

1.2.2 CM Environment

This section outlines the hardware and software platforms used in the project. Table 1 details the development environment on the servers used in this project. The server used for storing Configuration Items is a Solaris 2.6 machine and uses a TCP/IP protocol to connect to the code library. The machine that serves the project repository, located at collaborate.mit.edu/1.120.html, and future CM web site is running of a Microsoft Internet Information Server.

Table 1. **Development Environment on Server for ieCollab project**

Hardware	
Domain	mit.edu
IP Address	18.58.0.164, 18.58.2.155
Operating System	Solaris 2.6, WinNT
Connectivity	TCP/IP, HTTP
Web Management	N/A, Active Server Pages

Table 2 details the development environment on client machines. Programmers will work in both UNIX and WindowsNT environments on their own directories, using JAVA and CORBA development platforms.

Table 2. Development Environment on Client Machines

Hardware	
Operating System	UNIX, WindowsNT
Directory	multiple directories
Development Platform	JAVA, CORBA

1.3 Terms and Abbreviations

The following acronyms appear within the text of this standard:

CCB	Change Control Board
CI	Configuration Item
CM	Configuration Management
KM	Knowledge Management
PM	Project Management
QA	Quality Assurance
TE	Testers
PE	Programmers
RA	Requirements Analysts
CVS	"Concurrent Versions System"

2. Configuration Management

2.1 CM Management

The responsibilities of the CM team include configuration identification, configuration control, configuration status accounting, configuration audits and reviews, interface control, and change control.

2.1.1 Organization

The CM team is organized and located as shown in Table 3. However, though the CM team is allocated specifically for configuration management tasks, all members of ieCollab team are active participants in the Software Configuration Management plan; their adherence to the CM plan is essential to the success of the project and to the organization.

Table 3. CM Team organization

Title	Name	Location
Team Leader	Teresa Liu	MIT
Team Member	Anup Mantena	MIT
Team Member	Manuel Alba	CICESE

Although the CM team is an independent team within the ieCollab project structure, it will work very closely with other teams. These teams in particular are:

Change Control Board (CCB)
Knowledge Management
Project Management
Quality Assurance

The CCB, though a separate entity from the CM team, is an essential part of the Configuration Management process. The members of the CCB will include the team leaders of all teams in the project.

Table 4. Composition of Change Control Board (CCB)

Team	Team Leader
Project Manager	Joao- MIT
Business Manager	Justin-MIT
Marketing Manager	Steve-MIT
Requirement Analyst	Bharath-MIT
Designer	Hao-MIT
Programmer	Gyanesh-MIT
Quality Assurance	Nhi-MIT
Tester	Kenward-MIT
Configuration Manager	Teresa-MIT
Knowledge Manager	Paul-MIT

Through working with the CCB, CM will indirectly cooperate with all teams in the ieCollab project.

2.1.2 CM Responsibilities

The Configuration Management process consists of many activities, most of which are handled by the CM team. However, a few of these activities fall under the responsibility of other teams, such as the CCB, QA, and Testing teams. In addition, as any Configuration Item is subject to change, all teams will be expected to make the changes that have been requested and approved.

Table 5 lists Configuration Management activities and who is responsible for carrying these activities out.

Table 5. Configuration Management Activities and Responsible Parties

CM Activity	**Function**	**Responsible Team**
Identify CIs	Configuration Identification	CM
Version code CIs	Configuration Identification	CM
Name document CIs	Configuration Identification	CM
Store CI s	Configuration Identification	KM
Store Approved CIs	Configuration Identification	CM
Define Baselines	Configuration Identification	CM
Request Change	Configuration Control	Any Team
Evaluate Change	Configuration Control	CCB
Approve/Disapprove Change	Configuration Control	CCB
Notify appropriate team to make change	Configuration Control	CCB
Make requested change	Configuration Control	Team Owning CI
Review, audit the change	Configuration Control	CCB
Establish new baseline for testing after change	Configuration Control	CM
QA and or Testing activities on new baseline	Configuration Control	QA, Testers
Rebuild appropriate version of software	Configuration Control	CM
Distribute new version	Configuration Control	CM
Review changes to CIs	Configuration Audit and Review	CM
Track controlled CIs	Configuration Status Accounting	CM
Track/report status of requested changes	Configuration Status Accounting	CM

3. CM Activities

CM activities can be grouped into four functions: configuration identification, configuration control, status accounting, and configuration audits and reviews.

3.1 Configuration Identification

Controlled items will include source code, user documentation, data bases, test cases, test plans, specifications, management plans, and group presentations.

3.1.1 Identifying Configuration Items

The exact items to be tracked are listed in Table 6. These CIs will be stored in the appropriate repository, and this list may grow as the project progresses and new Configuration Items are identified.

Table 6. Configuration Items to be tracked by CM Team

Configuration Item Description	**Configuration Item Identifier**	**Author**	**Present status**
ieCollab Transaction Management Source Code	ieCollab-PR-Code-TM-?.java	PR Team	Available, In Progress
ieCollab Meeting Management Source Code	ieCollab-PR-Code-MM-?.java	PR Team	Available, In Progress
CAIRO Souce Code	ieCollab-PR-Code-CAIRO.java	PR Team	Available
Programming Team Presentation	ieCollab-PR-Presentation.ppt	PR Team	Available
RA Meeting Management Specifications V 1.4	ieCollab-RA-Spec-MM-1.4.doc	RA Team	Available
RA Transaction Management Specifications V1.6	ieCollab-RA-Spec-TM-1.6.doc	RA Team	Available
RA Presentation	ieCollab-RA-Presentation.ppt	RA Team	Available
DE Transaction Management Specifications V 1.0	ieCollab-DE-Spec-TM-1.0.doc	DE Team	Available
DE Meeting Management Specifications V 0.2	ieCollab-DE-Spec-MM-0.2.doc	DE Team	Available
DE Client Interface Specifications V 0.3	ieCollab-DE-Spec-CI-0.3.doc	DE Team	Available
DE Presentation	ieCollab-DE-Presentation.ppt	DE Team	Available
Testing Meeting Management Specification V 2.0	icCollab-TE-Spec-MM-2.0.doc	TE Team	Available
Testing Transaction Management Specification V 2.0	ieCollab-TE-Spec-TM-2.0.doc	TE Team	Available
Testing System and Integration Specification V 1.0	ieCollab-TE-Spec-SI-1.0.doc	TE Team	Available
Testing Reports	ieCollab-TE-Report?.doc	TE Team	Not Available
TE Presentation	ieCollab-TE-Presentation.ppt	TE Team	Available
QA Plan V 2.0	ieCollab-QA-Plan-2.0.doc	QA Team	Available

QA Presentation	ieCollab-QA-Presentation.ppt	QA Team	Available
CM Plan V. 1.0	ieCollab-CM-Plan-1.0.doc	CM Team	Available
CM Presentation	ieCollab-CM-Presentation.ppt	CM Team	Available
KM Plan	ieCollab-KM-Plan.doc	KM Team	Not Available
KM Presentation	ieCollab-KM-Presentation.ppt	KM Team	Not Available
User Manual	ieCollab-U_manual.doc	KM Team	Not Available

Project baselines will be defined at control points within the project life cycle. Events that create the project baselines are the reaching of a milestone in the project or a request to change the baseline. The project baselines can be divided into four categories: Functional, Allocated, Product, and Developmental Configuration baselines.

The baselines for the ieCollab project before reaching the first milestone are as shown in Table 7.

Table 7. Baselines for ieCollab project

Baseline	**Configuration Items in Baseline**
Functional Baseline	Business/Marketing Manager documents
Allocated Baseline	Requirement Analyst documents
Developmental Configuration Baseline	Design documents Project Manager Plan Testing documents
Product Baseline	We are unable to determine product baseline at this point.

The product baseline is generally determined at completion of product and delivered formally to the customer; thus, it is not possible to determine the product baseline at this time.

Aside from the evolution of the baseline from milestone to milestone, a baseline can be changed through the change control process. Any change request that is submitted, if approved by the CCB, can result in the changing of the baseline.

3.1.2 Naming Configuration Items

Naming configuration items encompasses both naming conventions and version marking of CIs. The naming convention of the CIs will be handled in a few different ways. According to the Programming Standards, program files will be named according to the class name. Other documents will be named according to the following standard:

project-team_name-description-version_number.extension

For example, the Requirement Analyst Specification for Meeting Management V 1.0 would be expressed as *ieCollab-RA-Spec-MM-1.0.doc.*

Version marking of the source code will be done according to the configuration management software tool we will use, Concurrent Versions System (CVS). CVS automatically assigns a version number to code that has been checked in and changed. After every milestone, a new version will be created manually by CM team.

3.1.3 Acquiring Configuration Items

This project has two controlled software libraries at this point in the project. The first library is the Code Library, which includes all source code for the project, including intermediate software development products. This library is housed on a server and controlled using CVS. User access will be authorized by PM but will include programmers, testers, QA, and CM team at the very least. Privileges granted, for example, for software will include check-in and check-out.

The other controlled software library will be called the Document Repository (DR), and it is the controlled collection of non-code documentation. The collaborate.mit.edu web repository currently serves as the DR; however, in the future, CM team's own web site will serve as the DR. The DR will house only approved documents and specifications, as well as the latest versions of each team's documents. Unlike the collaborate.mit.edu web site that provides an evolutionary view of the project documents, the CM web site serving as DR will house all documents that are Configuration Items and provide a static snapshot of CIs.

To begin with, all source code will reside on our server and be accessible using CVS. This code includes CAIRO source code, as well as all code produced by the Programming Team. As new versions and files are created, they will also be added to the server using CVS. Users will be able to check out files to make changes and check the files back in to update these changes. This process will be closely monitored by CM. Authorization by the relevant groups will be required before any changes can be made to the files in the repository, and only those with login names and passwords will be allowed to access the system for check in and check out.

All other documents such as standards, plans, specifications, etc. will be available through the project repository, located at collaborate.mit.edu/1.120.html or the future CM web site. All members of the project team will be able to read and download documents off the DR. In the case of the CM web site, which holds links to only approved configuration items, uploading is allowed only by the CM team.

3.2 Configuration Control

Members of the project team may request changes, encompassing both error correction and enhancement. The change process consists of the following steps:

1) Identification and documentation of the need for a change
2) Analysis and evaluation of a change request
3) Approval or disapproval of a request

4) Verification, implementation, and release of a change

3.2.1 Requesting Changes

Changes will be identified by change requestors, submitted to CM, and recorded by CM. When requesting a change, the requestor should provide the following information, by the way of a change request form, which will be recorded by CM:

1) The name(s) and version(s) of the CIs where the problem appears
2) Originator's name and organization
3) Date of request
4) Indication of urgency
5) Classification of the problem
6) Description of the requested change

CM will assign a change request number to each change request, and changes will be classified as described in Table 8. See the Appendix for a Change Request template.

Table 8. Categories used for classifying problems in software products

Category	**Applies to problems in:**
Plans	One of the plans developed for the project
Concept	The operational concept
Requirements	The system or software requirements
Design	The design of the system or software
Code	The software code
Test information	Test plans, test descriptions, or test reports
Manuals	The use, operator, or support manuals
Other	Other software problems

3.2.2 Evaluating Changes

CM will forward the change request to the Change Control Board (CCB) for evaluation of technical feasibility and analysis of the change request. The members of the CCB will consist of the team leader from each group and be lead by the Project Manager.

3.2.3 Approving or Disapproving Changes

The CCB will provide the final decision on whether the request is approved or disapproved Changes will be evaluated according to their effect on the deliverable and their impact on project resources. The Board will approve or disapprove the request and return a Change Authorization form if the change is approved.

3.2.4 Implementing Changes

After approval of a change request, CCB will notify the group that the change request affects using a Change Authorization Form. Once the requested change is complete, the CCB will verify that the changes made reflect the changes requested.

The information recorded by CM for the completion of a change shall contain the following as a minimum (with more added later):

1) The associated change request(s)
2) The names and versions of the affected items
3) Verification date and responsible party
4) Release or installation date and responsible party
5) The identifier of the new version

CM will then establish a new baseline for testing, and QA and/or Testing group will then perform a review and audit of the baseline software. Once the changed version passes review and audit, CM uses it to rebuild the appropriate version of the software and release it as a new version.

3.3 Configuration Status Accounting

This section describes the process used to provide configuration status accounting (CSA). CSA is the recording and reporting of information needed to manage configuration items effectively, including the items listed below.

To handle CSA, CM plans to utilize the project's repository, collaborate.mit.edu web site. Although it is often useful to automate the CSA process, we have been unable to come up with a feasible software solution.

Input data includes CCB decisions, such as approving or disapproving change requests, establishing configuration baselines, and approving the release of software for distribution. Input data also includes status information of CIs and change requests. Output data is formatted as CSA reports. These status reports will be produced when requested by PM, although CM plans to submit them once every two weeks.

The first type of report generated is the record of the approved configuration documentation and identification numbers and their overall status. Another type of report will be a report of the status of change requests and the implementation status of approved changes. A final type of CSA reports would be a report stating the results of a configuration audit of a changed CI.

3.4 Configuration Audits and Reviews

Configuration Audits will occur, at a minimum, before a new version of a CI is released. They determine to what extent the actual CI reflects the required physical and functional characteristics. After this approval occurs, CM can release the new version of the CI. See the Appendix for an example.

4. CM Schedules

CM works during the entire life cycle of the project, although most work is done during the programming stage. At this point it is difficult to determine the schedule of work since the programming schedule has not been set. However, since CM work is continual, it is not entirely critical to have the programming schedule in place.

5. CM Resources

CM resources are listed in Table 9. Our main tools are CVS and the Microsoft Internet Information Server, as well as the Active Server Projects technology. Training and research were necessary in learning about CVS.

Table 9. CM Tools/Resources

Tool	Location	Type of File Stored
CVS	cee-ta.mit.edu	source files
Microsoft Internet Information Server	collaborate.mit.edu	.doc, .ppt, .pdf
Active Server Projects	collaborate.mit.edu	.doc

6. Conclusion

Configuration Management is a very critical part of the project life cycle. Done correctly and with the help of all members of the team, it can mean the success of the project.

Appendix

Software Change Request

This section to be used by Configuration Management Team only		
Change No:	Date of Initiation:	Page of
Change Name:		

Submitted by (Name, Team):
Date of Submission:

Reason For Change:	1. Requirement Change	2. Design Change
	3. Error Discovered	4. Quality Assurance

Priority of the change HIGH ( ) MEDIUM ( ) LOW ( )

Change Description:

CIs Involved in Change

Cost of Evaluation: hours			Cost to Implement: days hours					
CI Identifier	Documentation Affected							
	Plans	Concept	Req	Design	Code	Test Info	Manuals	Other

Comments or Alternative Solution:

Proposed date of implementation :

CCB Decision:	Request Denied	Date:
	To be Evaluated	Name:
	Request Frozen	Signature:

Comments:

Software Change Authorization

This section to be used by Configuration Management Team only		
Change No:	Change Name:	Page of
Form No:	Date of Authorization:	

Changes To be Implemented

No.	Responsible Team	Date Due	SCN Date*	SCN Form No.*

* filled in when Software Change Notice (SCN) is completed

CIs involved in Change

CI Identifier	Documentation Affected							
	Plans	Concept	Req	Design	Code	Test Info	Manuals	Other

Comments:

Approved By:	Date:
SCCB Approval Name:	Date:
Signature:	

Software Change Notice

This section to be used by Configuration Management Team only		
Change No:	Change Name:	Page of
Software Change Notice No:	Processing Date:	

Submitted by (Name, Team):
Date of Submission:

Changes Implemented

No.	Implementation Status	Responsible Team	Date

Configuration Item Revision Changes

Configuration Item Description	Configuration Item Identifier Prior to Change	Configuration Item Identifier After Change

Comments

Approved By:	Date:
Title:	Signature:

CCB Approval:	Name:	Date:
	Title:	Signature:

Received By CM:	Name:	Date:

Configuration Audit

Name of auditor:______________________________________

Date of the audit:______________________________________

Please ask and answer the following questions.

1. Have changes specified in the Software Change Request or any other forms been made or considered?

YES / NO
comments:____

2. Have formal technical reviews been conducted to achieve technical correctness?

YES / NO
comments:____

3. Have comments or suggestions from formal technical reviews been considered or incorporated?

YES / NO
comments:____

4. Have software engineering standards been properly followed?

YES / NO
comments:____

5. Have CM procedures for documenting changes been followed?

YES / NO
comments:____

6. Have all related CIs been properly updated?

YES / NO
comments:

7. Any additional comments?