Requirement Analysis for Distributed Software Development

by

Yu Lung Alan Ng

B.S.E., Civil and Environmental Engineering (1999)

University of Michigan at Ann Arbor

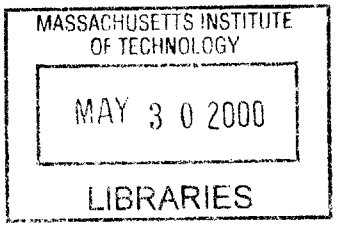
ENG

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

June 2000



© 2000 Massachusetts Institute of Technology
All rights reserved

Author.....
Department of Civil and Environmental Engineering
May 18, 2000

Certified by.....
Feniosky Peña-Mora
Associate Professor of Civil and Environmental Engineering
Thesis Supervisor

Accepted by.....
Daniele Veneziano
Chairman, Departmental Committee on Graduate Studies

Requirement Analysis for Distributed Software Development

by

Yu Lung Alan Ng

Submitted to the Department of Civil and Environmental Engineering
on May 5, 2000,

in partial fulfillment of the requirements for the degree of
Master of Engineering in Civil and Environmental Engineering

ABSTRACT

The software engineering industry has always been faced with different challenges on developing larger and more complex software with higher quality, shorter time, and lower cost. To overcome these challenges, software development organizations have adopted the use of distributed software development teams as a popular practice to improve the quality and productivity in recent years.

However, the lack of social and human interaction skills needed for collaboration, which are essential in the requirement analysis process between clients and developers, hinders successful implementation of the team-based distributed software development practice. While software developers often recognize the collaboration problems in the critical distributed requirement analysis process, they typically cannot identify or implement effective solutions.

This thesis provides the detailed examination of requirement analysis for distributed software development and addresses the collaboration issues through the comprehensive discussion of a case study. The results of the case study show that if good collaboration, coordination, and cooperation are not exercised during the project, problems will appear that can lead to project failure. Also, attention must be paid to the following four elements in order to have a successful requirement analysis for distributed software development: people, product, process, and technology.

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

Acknowledgements

First of all, I would like to thank Professor Feniosky Peña-Mora for his guidance and advice on this thesis. I would also like to thank him for his teaching and support on the ieCollab project.

Next, I want to express my appreciation to all the ieCollab team members, especially the requirement analysis team members. I will never forget the tears and joy that we have shared through the project.

Also, I want to acknowledge the support from all the Master of Engineering students, especially those in the Information Technology track. Our countless time and effort that we spent on all the assignments and studies in MEng room, Athena clusters, and Design Studio of the Future makes my MIT experience an unforgettable one.

I would like to recognize my dad, mom, brother, and the rest of my family, who have always given me abundant support and love throughout my life, and have done so again during my studies at MIT. I would like to express my deepest gratitude to all of them.

Lastly, I would like to say a word of thanks to my beloved girlfriend Carol, who always supports me in whatever I do. She has given me the greatest encouragement throughout my tribulations at MIT. Without her, this thesis and my study at MIT would not be a success.

Contents

List of Figures

List of Tables

Chapter One

Introduction

This thesis examines the process of requirement analysis for distributed software development. It uses as a case study a classroom experience on the whole software development process through a nine-month project. The case study helps to understand some of the practical procedures of requirement analysis in software engineering, as well as to help experience the developmental problems on such environment in order to provide some recommendations and future work in this field.

1.1 Motivation

The context of software engineering has changed continuously ever since its first existence about four decades ago. Although a great deal of wisdom has been acquired regarding the proper methods of software engineering, it remains difficult to apply the knowledge in changing situations for different software engineering projects.

One of the recent changes in software engineering is the emergence of the distributed development environment. Collaboration is certainly one of the frequent obstacles that software engineers need to face during the software development process. In a distributed software development environment where teams are physically located in

different places, collaboration becomes one of the key factors that can determine the success of the project.

Requirement analysis is one of the first processes for software development, and it is also possibly the hardest part of building a successful software. The complexity of the requirement analysis process increases when the software is developed in a distributed environment. The changes and problems encountered will be particularly important to understand and solve in order to cope with the rapidly changing software development process. The experience gained from dealing with these problems will also help the teams involved in the later software development processes.

As a result, requirement analysis for distributed software development becomes one of the most important concepts to learn and study.

1.2 Thesis Overview

There is a total of six chapters in this thesis. The following paragraphs are brief descriptions on each of them.

Chapter One is simply the introduction. In this chapter, the motivation to write this thesis is presented, then a brief overview of the structure of this thesis is provided.

In order to study and comprehend the importance of requirement analysis, one must first understand the major principles and practices of software engineering. Chapter Two provides this background. This chapter first covers the history behind software engineering, with a look at the different software development processes, approaches, and environments.

In Chapter Three, the generic approach for requirement analysis is first introduced with detailed discussion on different activities in the process, followed by the distributed development approach with the specific needs and tools required for requirement analysis in a distributed development environment.

In Chapter Four, the distributed software engineering project – ieCollab is described as a case study in a detailed manner. A brief project description is first given, followed by project and team organizations. Then, requirement analysis in ieCollab is presented. Finally, problems faced during the project and recommendations for improvement are given.

Finally, Chapter Five is the conclusion, which presents the successful factors and future work of distributed requirement analysis based on the findings in this thesis.

Chapter Two

Software Engineering

To learn the development processes of software engineering, the definition of "software engineering" must be agreed upon. According to the Institute of Electrical and Electronics Engineers (IEEE), software engineering is defined as follows (IEEE, 1983):

(1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.
(2) The study of approaches as in (1).

This chapter will briefly introduce the background of this unique engineering, its many changes over the years and the improvements that it has gone through in different development processes, approaches and environments.

2.1 Background

Since the evolution of software engineering, people have had long debates on the definition of software engineering and on the correct approach to apply this engineering to real world problems. It can be said that the change in software engineering will always

be an on-going process, and that it will be particularly difficult to define and apply to the infinite variety of software projects. Each distinctive project needs a unique method of software engineering, therefore exists so many dissimilarities and changes in this field.

2.1.1 Origin

In 1968, software engineering first emerged as a popular term in a North Atlantic Treaty Organization (NATO) conference held in Garmisch, Germany. It was held to address the problems of building software applications and to improve productivity and reliability of software development. At that conference, NATO established the concept of software engineering. It also identified a set of issues and provided a foundation for the investigation of potential solutions (Blum, 1992).

The definition of software engineering proposed by Professor Friedrich Bauer at the conference still serves as a basis for discussion: Software engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines (Blum, 1992).

2.1.2 Modern Views

Many people view software engineering as a discipline of resolving problems with software solutions and producing software solutions to real problems nowadays. The basic objective of software engineering is to develop methods and procedures for software development that can be scaled up for large systems and that can be used to consistently produce high quality software at low cost with short cycle time.

However, there is no particular answer to the definition of software engineering. The answer depends on the problem being solved and the tools available in the software engineer's environment. There is no consensus regarding which are the best tools and methods. The main goal is to produce a successful software project, which delivers its planned products within schedule and budget and meets its defined functional and quality requirements.

2.2 Software Development Process

The path of software development has been extremely costly. When development is done incorrectly, inefficiently, or without planning, millions of dollars can be wasted. For example, the Government Accounting Office (GAO) has published a study on a software development project costing approximately 6.8 million dollars. This study calculated that out of the 6.8 million dollars spent, only $119,000 was spent in software development efforts that could be used as delivered. The rest of the efforts, worth 5.1 million dollars and approximately 75% of the total price tag, produced software that could not be used at all (Medina Sanchez, 1993).

There are many other similar statistics that show how many times software development efforts were wasted in the past. These statistics and examples emphasize the importance of building up and learning effective software development in order to produce software on time and within budget.

2.2.1 Software Process Framework

A common software process can be viewed and characterized as a simple framework as shown in Figure 2-1. It is not a set definition of the processes that an organization must

follow. It is only a reference model or guide for a software development process (Pressman, 1997).

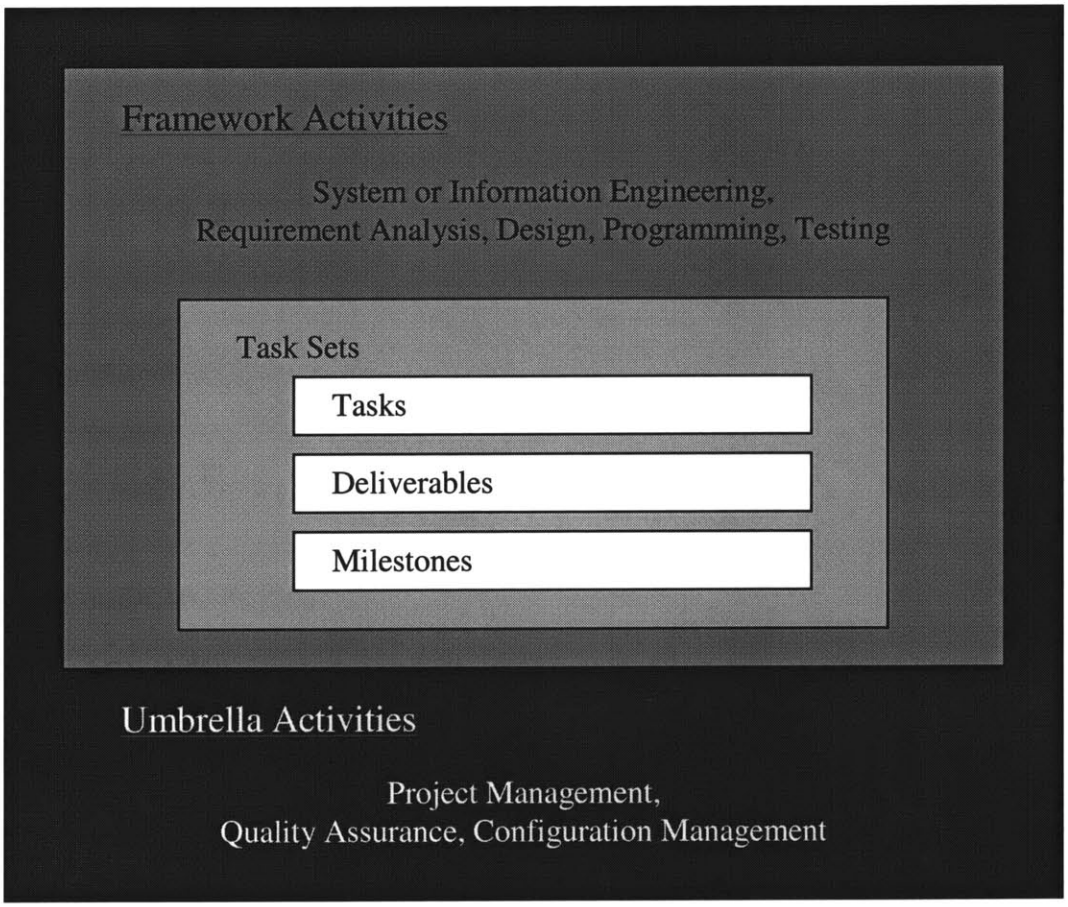


Figure 2-1: The Common Software Process Framework (Pressman, 1997)

The framework is composed of framework activities and umbrella activities. Framework activities are common phased procedures that are applicable to all software projects while umbrella activities are independent of any framework activity and occur throughout the process.

2.2.1.1 Framework Activities

In framework activities, there are task sets which are collections of work tasks, comprised of deliverables and milestones. Regardless of the software project size and complexity, the work in framework activities can be divided to three generic phases: definition, development, and maintenance. A phased development approach is used to better manage the development process and to achieve consistency. Each phase consists of a set of activities to accomplish the goals of that phase and each phase ends with a defined output.

Definition Phase:

The definition phase focuses on what the software should be. The key requirements of the system and software are defined. Two major tasks will usually occur in this phase: System or Information Engineering and Requirement Analysis. The followings are concise explanations of each task:

System or Information Engineering: The system engineering process is called information engineering when the context of the engineering work focuses on a business enterprise. It focuses on issues that happen before the start of the software engineering process such as the analysis of the whole system.

Business Management: It is a branch of Information Engineering that mainly responsible for defining the project goals, writing business plan and working on other business issues of the software development process.

Marketing Management: It is another branch of Information Engineering that is responsible for performing marketing research to understand customer's needs in the market. The information will be used to gather the business ideas of the software.

Requirement Analysis: The purpose of requirement analysis is to understand the problem the software is to solve. The emphasis in requirement analysis is on identifying what is needed from the system, not how the system will achieve its goals. Requirement analysts have to bridge the communication gap between the customer and the developer and to understand the problem clearly in order to correctly define all the requirements in the software requirement specification document.

Development Phase:

The development phase focuses on how the software should be built. The major development processes of the software are accomplished. Three specific technical tasks should always occur in this phase: design, programming, and testing, explained below:

Design: The purpose of the design phase is to plan a solution of the problem specified by the requirement specification document. In contrast to the requirement analysis, design starts the process on how to satisfy the needs instead of specifying what is needed. Designers have to communicate frequently with requirement analysts in order to produce the desired software design for the design document.

Programming: The goal of the programming phase is to translate the design of the system into programming codes. The aim in this phase is to implement the design using programs that are easy to read and understand. Simplicity and clarity should be emphasized to ease the task of testing afterwards. Programmers should keep in close contact with designers to correctly understand the design and deliver the final codes.

Testing: The goal of testing is to uncover requirement, design, and coding errors in the programs. Different levels of testing are used to detect errors in the software. Test plans with expected results are produced for different test units. Testers then run the tests and

compare the actual results with the expected ones. They have to produce the final test report and error report by the end of the testing phase to make sure the software works in the planned way.

Maintenance Phase:

The maintenance phase focuses on the change of the software. The maintenance phase reapplies the steps of the definition and development phases for the existing software instead of creating a new one.

Each of the framework activities mentioned above has its own task set at the beginning of the project and needs to produce certain deliverables by the end of its process. Setting milestones, which designate when certain tasks and deliverables are ready, monitors each phase.

2.2.1.2 Umbrella Activities

Umbrella activities overlay the whole process model and are applied throughout the software process. It complements the framework activities through the work of Project Management, Quality Assurance, Configuration Management, and Knowledge Management Teams. The following provides the definitions and explanation of these groups:

Project Management: The project planning task is done by the project management team. The team is responsible for project organization, planning, monitoring and control in order to deliver a quality product.

Quality Assurance: The purpose of quality assurance is to ensure all the software development processes are done at the right time and in the right manner. Quality assurance engineers are responsible for making sure that the software conforms to all the requirements, standards, and characteristics that are expected of professionally developed software.

Configuration Management: The goal of configuration management is to maximize productivity by minimizing mistakes because change and confusion often arise as the computer software is built. Configuration managers are responsible for identifying the changes, controlling the changes, making sure it is properly implemented, and reporting the change.

Knowledge Management: Although knowledge management is not regarded as one of the common software processes, its importance has grown over the years. The main function of knowledge management is to handle all the documents from the beginning till the end of the project. Knowledge managers are responsible to maintain all the documents for easy retrieval and storage.

2.2.2 Traditional Software Process Model

A software process model for software development specifies the activities in the software process framework that should be performed and the order in which they should be done. The model is chosen based on the nature of the project and application, the methods and tools to be used, and the controls and deliverables that are required. One of oldest and the most widely used models for various software projects nowadays is called the waterfall model as shown in Figure 2-2 (Pressman, 1997).

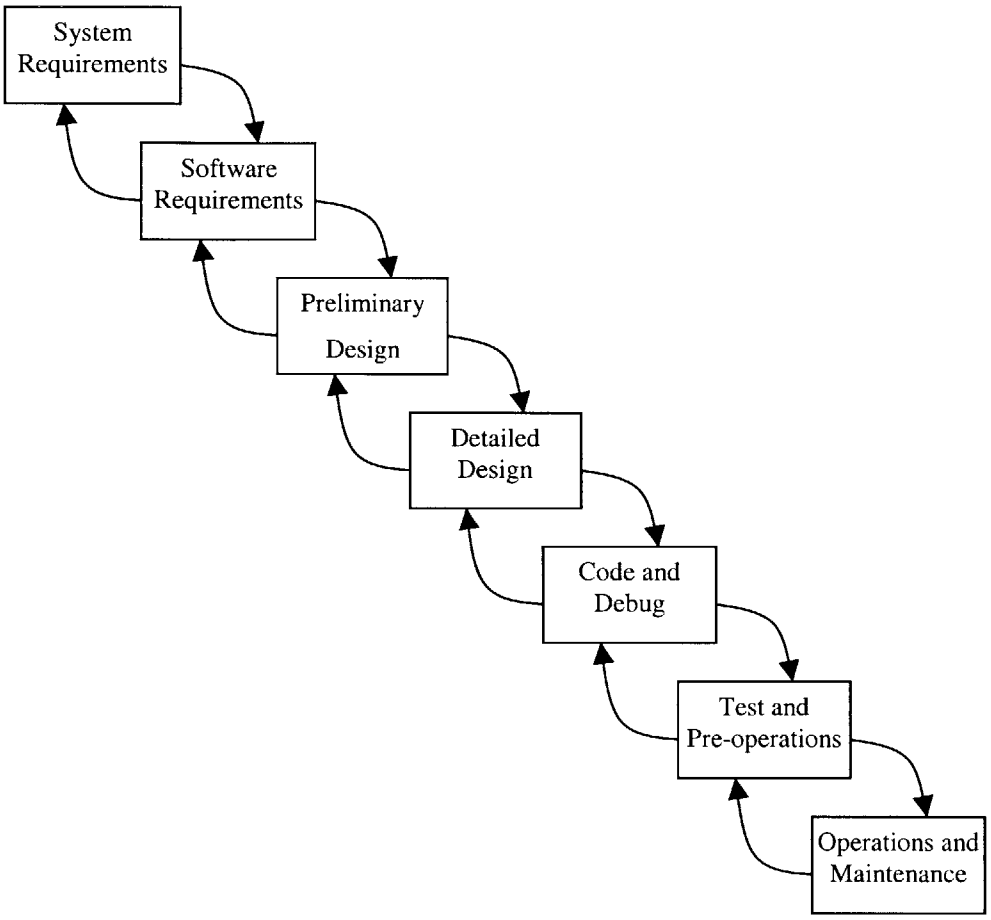


Figure 2-2: The Waterfall Model (Pressman, 1997)

Waterfall Model:

The waterfall model is also referred to as the linear sequential model or the classic life cycle. It was first defined as early as 1970 by Dr. Winston Royce to help cope with the growing complexity of the aerospace software products being tackled (Dorfman et al, 1997). With several years of experience with developing software for spacecraft mission planning, commanding and post-mission analysis, Dr. Royce had experienced different degrees of success. The resulting sequence of steps that he outlined in the model, with

various refinements and minor modifications, became the basis for the software development process over the last two decades (Dorfman et al, 1997).

The waterfall model is the simplest software process model in which the development phases are organized in a linear order. It uses a systematic, sequential approach and provides a template for software development that begins at the system level and progresses through requirement analysis, design, coding, and testing.

There are certain advantages of waterfall model because of its linear ordering of activities. It clearly identifies the end of each phase and the beginning of the next. Since each phase has defined its input and output, the waterfall model provides a certification mechanism that can be used to check and evaluate the consistency of the product produced by the end of each phase. It also makes the development process more structured and manageable because project managers can track the progress easily and more accurately to uncover possible delays (Jalote, 1997).

On the other hand, there are certain limitations of waterfall model. At the beginning stage of many projects, it is often difficult for the customer to state all the requirements explicitly. The waterfall model requires this and has difficulty accommodating this natural uncertainty. Because of the uncertainty and possible delays throughout the project, it may lead to inefficient time periods in which some project team members must wait for other members of the team to complete dependent tasks before beginning theirs. Besides, real projects sometimes do not follow the sequential flow that the model proposes. This may cause confusion if changes and alternations have to be made (Jalote, 1997).

Since requirement analysis is the first process in the waterfall model, the requirement analysis team does not have to wait for other teams' work before the analysis process

begins. However, any possible delays in this process will directly affect the time schedule of the whole software development process. As a result, requirement management must be carefully practiced to avoid the potential time pressure from other development teams.

In the distributed development environment where developers are in different locations, collaboration between requirement analysts and clients, and collaboration among requirement analysts in different locations are particularly difficult and time-consuming. Therefore, in typical software projects which demand rapid development, the linear and sequential waterfall model may not be a proper choice.

Overall, the waterfall model is still well suited for routine types of projects where the requirements are well understood. If the developers are familiar with the problem domain and the requirements for the software are quite clear, the waterfall model works well.

2.2.3 Evolutionary Software Process Model

Software usually evolves over time and the requirements often change as the software development process proceeds. However, the traditional software process models, like the waterfall model, do not consider the evolutionary nature of the software. As a result, evolutionary software process models with an iterative nature have become more popular, helping software engineers to develop more complete versions of the software (Pressman, 1997).

Incremental Model:

The incremental model is one of the popular evolutionary software process models. It applies the linear sequences in a staggered fashion as shown in the Figure 2-3.

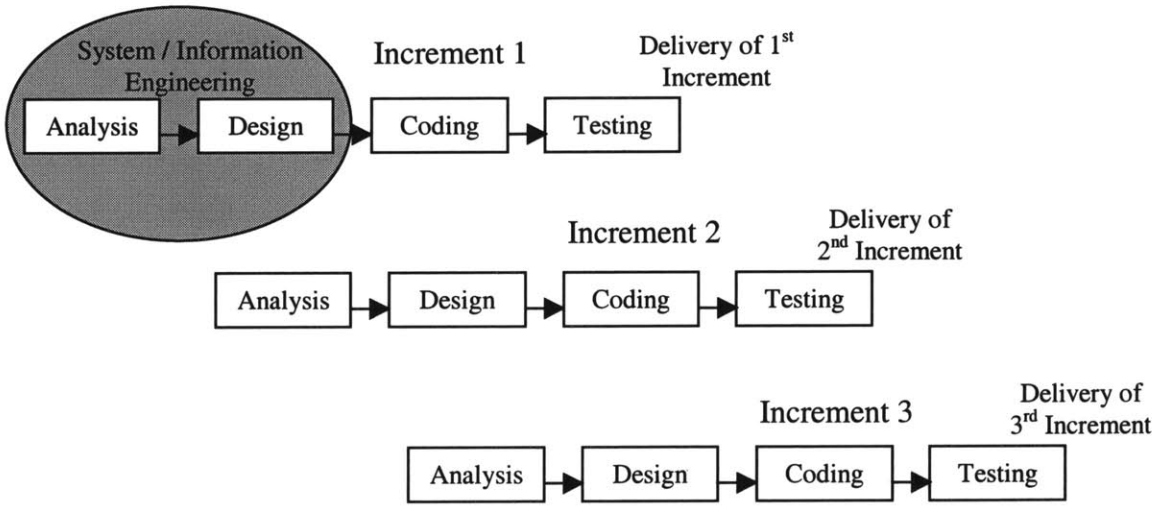


Figure 2-3: The Incremental Model (Pressman, 1997)

After each sequence, a deliverable version of the software should be available. These deliverable versions are called increments. The first increment is called the core product. Basic requirements of the software are addressed in the core product while other supplementary requirements are addressed in the later increments. Each increment is reviewed by clients and the evaluations will be used as a basis to improve the quality of the software. Any additions or modifications of the requirements will be addressed in the later increments. These processes will be repeated until the final complete software is produced (Pressman, 1997).

There are certain advantages of the incremental model because of its iterative nature. The incremental model is particularly useful when there is a shortage of developers in different processes of software development. The developers do not have to deliver the final version of the product in each process of the increment. Therefore, fewer numbers of people are needed. Also, the developers in the next process do not have to wait until the earlier process finishes and delivers the final version in order for them to start their work. Therefore, it saves time and resources for the development process. In addition, since the model allows frequent feedback from the clients, the quality of the software can improve significantly (Pressman, 1997).

On the other hand, there are certain pitfalls of the incremental model. The overall quality or long-term maintainability of the software may not be considered when each deliverable version is developed for the clients in a short period of time (Pressman, 1997). Also, the iterative nature of incremental model makes project managers more difficult to allocate resources and plan the schedule.

For the requirement analysis process, the incremental development nature of the incremental model helps to get more feedback from clients. This is particularly useful for requirement analysis because requirements are gathered from the clients and their active participation and collaboration between requirement analysts and clients is essential. Also, since the design phase starts after the core product is produced, feedback from the designers' point of view also helps to improve the requirements in the next version of the software.

In the distributed development environment which collaboration is the key success factor, the incremental model enhances the collaboration process by the feedback mechanism and the iterative nature of the model. The participation from different development teams

on developing increments further encourages the involvement and collaboration between distributed team members early in the beginning of the project.

Overall, incremental model increases the flexibility of the software process as well as the management complexity of the software project.

Spiral Model:

The spiral model is another popular evolutionary software process model. It applies both the iterative and systematic nature of other process models with processes moving around a spiral as shown in the Figure 2-4.

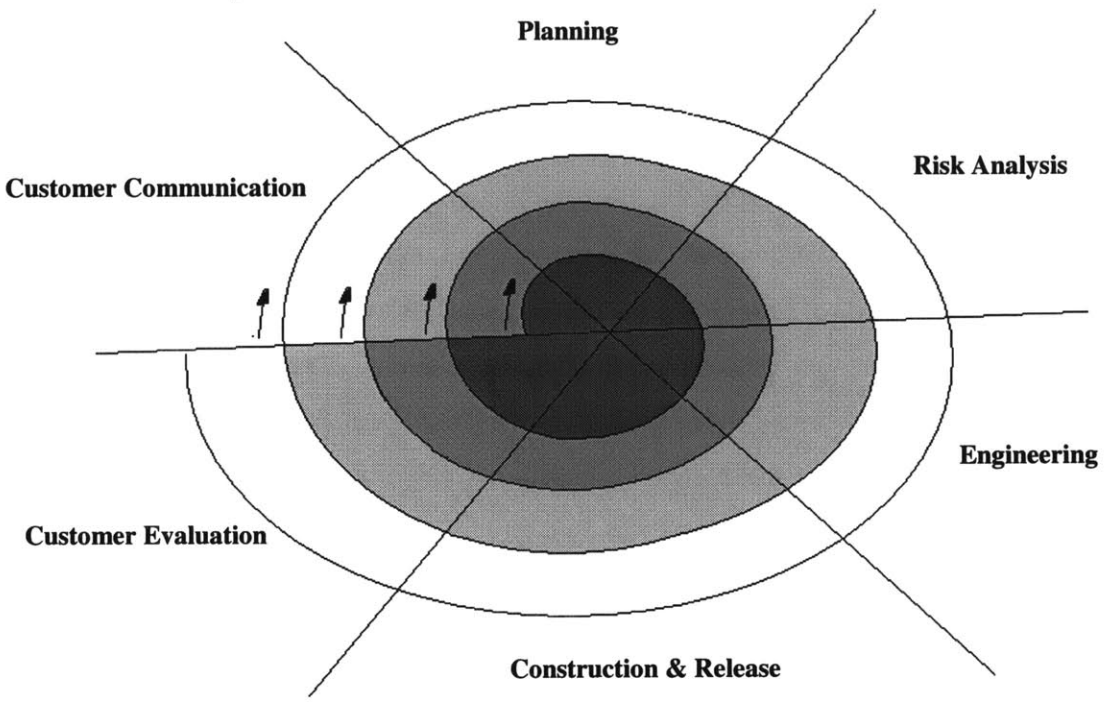


Figure 2-4: The Spiral Model (Pressman, 1997)

The spiral model is divided into six task regions and one iteration of the following six tasks should provide a more complete version of the software than the previous one (Pressman, 1997):

- *Customer communication*: it is used to establish effective communication between software developers and customers.
- *Planning*: it is used to define the project schedule and resources.
- *Risk Analysis*: it is used to evaluate managerial and technical risks.
- *Engineering*: it is used to build the applications' representations.
- *Construction and Release*: it is used to construct, test, install and provide user support such as documentation and training.
- *Customer Evaluation*: it is used to obtain customer feedback from the evaluations.

The software process begins at the core and moves around the spiral in the clockwise direction as the process continues. It starts with the customer communication task and then project plan can be adjusted accordingly in the planning task region. Then, the risk analysis is done after any changes in the project plan. After that, engineering and construction and release tasks are accomplished for the release of the software. Finally, customer feedback is got from the customer evaluation task region. One version of the software is released after each cycle and then whole process is repeated until the complete version is developed (Pressman, 1997).

The most important advantage of the spiral model is the detection of risks at early stages of software development. As the software development process proceeds, both the developers and the customers have a better understanding of what they want and are able to react to risks at different levels. Also, it has risk analysis and customer evaluation task regions in each cycle, which help to uncover and reduce the risk by detecting them early in the process (Pressman, 1997).

On the other hand, the disadvantage of the spiral model is that it is complicated. It generally requires more advanced management skills and sometimes it may be hard to define objectives and milestones that indicate whether it is ready to move on to the next cycle or not. Besides, some projects may not have a lot of risks and do not need to have so many risk analysis and customer evaluations. These may become redundant work if the development process is straightforward and risks are modest (McConnell, 1996).

For the requirement analysis process, the nature of the spiral model helps to build the communication between customers and developers and get more feedback from customers. Like the incremental model, this is particularly useful for requirement analysis because requirements are gathered from the customers and the collaboration between customers and developers is very important.

In a distributed development environment, the planning and risk analysis task processes are useful for project re-scheduling and analysis of various risks, which are generally needed for the tight schedule and high risk distributed environment. Also, the customer communication and customer evaluation task processes are essential for collaboration, which is the most important process in a distributed requirement analysis process.

Overall, the spiral model is good for projects that have knowledgeable management practice because it requires more management skills to monitor all the risk analysis and customer involvement processes than other models.

2.2.4 Software Maturity Model - Capability Maturity Model

While the software process model describes the activities that should be performed throughout the software development process, many software organizations still faced

many quality and productivity problems during the last three decades. Because of this, in November 1986, the Software Engineering Institute (SEI) began to develop a process maturity model that would help organizations improve their software development process. Based on the software process assessments and feedback from both industry and the government, the SEI released the first version of the Capability Maturity Model for Software (CMM) in 1991 (Paulk and Curtis et al, 1993).

CMM provides a measure of the effectiveness, the state of process maturity and the capability of a company's software engineering practices by establishing five process maturity levels. Besides Level 1, each maturity level is decomposed into several key process areas that identify issues which an organization should focus on to achieve a maturity level. The CMM model with key process areas is shown in Figure 2-5 (Paulk and Curtis et al, 1993) (Paulk and Weber et al, 1993).

Each of the five levels in CMM has certain primary processes that need to be finished. Level 1 is the initial level in which the software development process is characterized as ad hoc and few processes are defined. The success depends on individual effort. Level 2 is the repeatable level in which the earlier successful processes can be used to repeat for projects with similar applications. Basic project management processes are established in this level. Level 3 is the defined level in which the software process for both management and engineering activities is documented and integrated into a standard software process for developing and maintaining the software. Level 4 is the managed level in which detailed measures of the software process and product quality are collected and controlled. Finally, Level 5 is the optimizing level in which quantitative feedback is received to continuously improve the software development process (Paulk and Curtis et al, 1993).

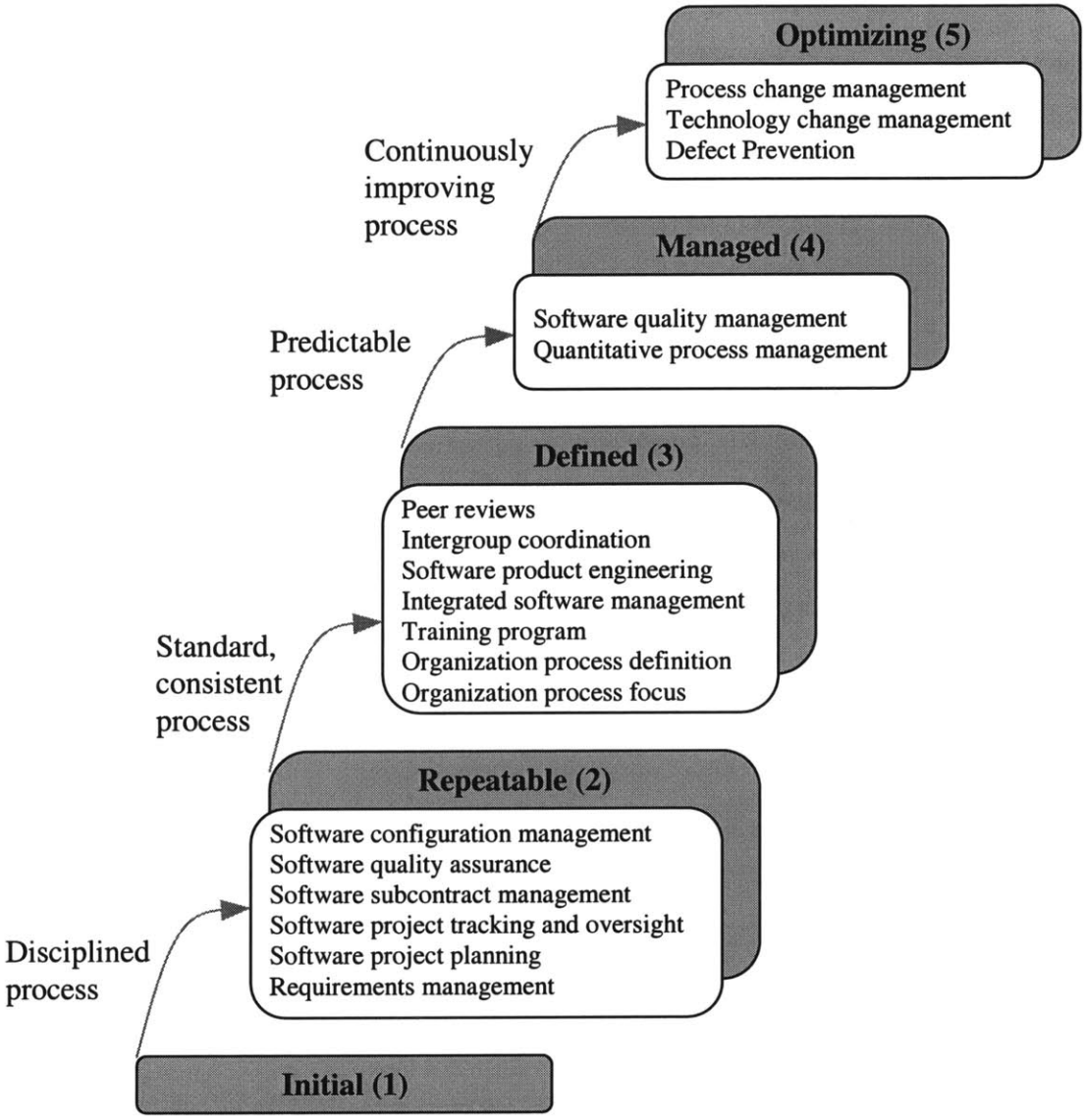


Figure 2-5: The Capability Maturity Model with Key Process Areas
(Paulk and Curtis et al, 1993) (Paulk and Weber et al, 1993)

Requirement analysis comes to the CMM in Level 2 as requirement management in one of the key process areas. The detailed discussion on this is in section 3.2.5.2. In Level 1, the requirement analysis process does not have formalized procedures or plans. Therefore, quality of the software requirement specification and the performance of the

team depend a lot on the skills of the requirement analysts. In Level 2, policies and procedures are implemented for requirement analysis based on past experience. Better management system is followed so earlier successes of the process can be repeated. In Level 3, the organization's standard procedures of the requirement analysis are documented and followed. The consistent practice helps to improve the effectiveness of the requirement analysis process. In Level 4, measurable limits are set for the requirement analysis process to ensure good quality and productivity. With these quantitative measurements, the requirement specifications have predicted high quality. In Level 5, continuous improvement can be done through the analysis of defects and the application of new technologies and methods in the requirement analysis process (Paulk and Curtis et al, 1993).

In a distributed software development environment, the distributed locations of different team members make the application of CMM harder to determine the maturity of the software organization. Policies, standard procedures, and measurable limits in different levels are more difficult for organizations to establish for distributed teams. In addition, the new technology for distributed requirement analysis process is even harder to develop.

Through the five levels in CMM, software development organizations can get guidance on how to manage and gain control on software developing processes. It also helps them to determine current process maturity and identify the most critical issues for process improvement so they can select the appropriate strategies to progress on their projects.

2.3 Software Development Approach and Environment

After the software development organization chooses which development process and model to use, there still remains two questions: with which approach and in what environment should they use to develop the software.

There are two commonly used software development approaches and two commonly used environments. The two approaches are object-oriented and structured approaches, which have been predominant in development approach. On the other hand, distributed and co-located environments are the two most widely used development environments. In this section, these approaches and environments are discussed in the context of general software engineering practice. Further discussions on how the process of requirement analysis applies on these approaches and environments will be presented in Sections 3.2.2 and 3.3 respectively.

2.3.1 Object-Oriented versus Structured Approach

The structured software development approach was based on functional decomposition or event partitioning. Although this approach offered a vast improvement on the development process for different projects in the late 1970s and early 1980s, it has since grown past maturity and exhibits significant limitations. The advantage of the structured approach is that it has a systematic approach for analyzing a problem by decomposing it into parts and describing the relationships between them. However, it was created for traditional structured programming languages such as FORTRAN and C and is limited to functional building blocks. The software developed using this approach tends to be both difficult to understand and expensive to develop and maintain. Also, it has limited potential for significant reuse. Moreover, the transitions between activities in the software development process are more difficult to maintain because structured

development paradigms tend not to be uniform and use different methods and notations during different activities such as requirement analysis and design (Firesmith, 1993) (Medina Sanchez, 1993).

On the other hand, the object-oriented approach is given to a set of related development methods based on the concept of an object. This object is an abstraction or model of a single application-domain entity that has structure, state, and behavior. Instead of being based on functional decomposition or event partitioning, the object-oriented approach analyzes, designs, implements, and tests software applications in terms of their component objects, classes of related objects, and the subassemblies and frameworks of related objects and classes. By using the object-oriented approach, one can ease the transitions between development activities by using the same paradigm, concepts and models. Also, it brings benefits in terms of code size, efficiency and reuse by using object-oriented programming languages such as C++ and JAVA. Moreover, it improves the extensibility, maintainability, and reusability by localizing around objects and classes which are more subject to change than around functions (Firesmith, 1993).

As a result, the object-oriented approach has proven to be the next great paradigm for software development in the future. A more detailed structured and object-oriented approach discussion on requirement analysis is presented in Section 3.2.2.

2.3.2 Distributed versus Co-located Environment

The defining difference between distributed and co-located environments are whether the software developers work in the same location or are spread across many different cities or even countries. Each environment has its own good and bad qualities. These attributes and their contributions to software development are explained in this section.

The co-located software development environment has long been used in different projects since software engineering first existed. It is probably the easiest way to start developing software, especially if the organization lacks software development experience. No particular communication technology is needed because all the people are located in the same place. Therefore, many software organizations have used co-located software development environment because it helps to save time for communication and money for acquiring advanced technology.

Since in a co-located environment, all software engineers work in the same place, they can talk and discuss the progress face to face. Therefore, problems can be solved by direct communication and collaboration problems can be minimized. In the past, software industry focused on developing small-scaled software system when technology was not very advanced and communication remained a problem. The co-located environment seemed to be a sufficient and efficient way for software development.

On the other hand, software development environment has significantly changed in the last decade and there is a shift towards geographically distributed software development. This shift is caused by the increasing competition in the software development industry and by the development of larger and more complex software.

The goal of the software industry is to provide better, more technologically advanced software for lower prices. Since the software development is labor-intensive, development of large-scaled software systems requires a large number of software engineers. Sometimes, it can be difficult to hire this large number of engineers, who specialize in a particular field, in one geographical location. Also, applications are becoming more complex and different phases in the software development process are performed across organizational boundaries (Aoyama 1990). Therefore, it is impossible

to allocate all engineers on one development site and they will tend to be more distributed and work in different locations.

Also, there are changes in the social environment induced by the global network and geographically distributed software development projects have been made possible by rapid development in the data communication area.

Many software companies have recognized that distributed collaboration has great potential for the near future and they have tried to incorporate the concept of distributed and collaborative software engineering into their current practice (Johansson et al, 1999). Also, they are looking toward collaborative teams to increase teamwork in order to improve quality and productivity in software development.

However, the gradual shift towards geographically distributed software development is not entirely recognized by the industry or supported by present tools. Many software companies still believe distributed development environment is not necessary for small organizations for developing small-scaled software systems (Haag et al, 1997). Also, although the current technology allows faster or even real-time communication between distributed locations, the availability and accessibility of current tool still remain problems in different development locations.

Besides technological problems, there are other problems with using the distributed software environment. In distributed locations, there may be time zone difference among the many groups and this problem cannot be overcome no matter how advanced the technology is. Also, cultural and social differences make the collaboration between people in different locations more difficult. It often takes longer time to solve the conflicts between people working in distributed locations than those working in co-located development environment.

There is continuous research all over the world to solve all the possible problems that distributed software engineers are facing now. Although distributed software development environment still has many problems, it seems inevitable that it will become widely used and recognized in the future when the scale of software and competition in the software industry become larger and larger.

2.4 Summary

As one can see, software engineering has been a rapidly changing area, with no detailed instructions on how the process should proceed. Although it has been established that all software projects require the phases of definition, development and maintenance, what methods to use for these steps are not definite and always changing.

However, the basic principles behind all these steps are always applicable to all software projects. Multiple options using these principles exist for every facet of software development, each with pros and cons. Thus, each software project must be evaluated before committing to a particular form of process modeling, choosing between object-oriented or the structured approach, and deciding upon which development environment fits the project. As the correct choices between different process models, approaches, and environments are the first step to a successful software development project, software engineers should never forget the importance of these basics and the principles behind these concepts.

Chapter Three

Requirement Analysis in Software Engineering

In society, not only in the past, but also the present, the general stereotype for software engineering would illicit this view: a group of engineers coding and debugging for hours on end in front of computers. To many people, the core of a software engineering project should be the creation of the actual software: the programming. Everything done prior and post programming, such as requirement analysis and software testing, is relatively unimportant.

Dr. Frederick Brooks, a professor at University of North California, thinks otherwise (Brooks, 1987). He defined requirement analysis as follows:

"The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements, including all the interfaces to people, to machines, and to other software systems. No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later.

Therefore, the most important function that the software builder performs for the client is the iterative extraction and refinement of the product requirements. For the truth is, the client does not know what he wants. The client usually does not know what questions

must be answered, and he has almost never thought of the problem in the detail necessary for specification." (Brooks, 1987)

In this chapter, the background of this critical step of software engineering will first be discussed, followed by detailed descriptions of general and distributed requirement analysis approaches.

3.1 Background

Since the beginning of the software engineering era, the ability to produce complete, correct and unambiguous software requirements remains one of the major causes of software failure. At the beginning of the 1990's, the software engineering industry began to realize the importance of the requirement analysis process in software development, and more emphasis and recognition have been placed on the field known as requirements engineering (Thayer et al, 1997).

Some of the recent changes have greatly improved the requirement analysis practice. There is now an International Symposium on Requirements Engineering held in odd-numbered years, and an International Conference on Requirements Engineering held in even-numbered years. Also, the number of new books and journals on requirements engineering has been increasing recently. Moreover, the Software Engineering Institute included requirements engineering as a key process area in their influential Capability Maturity Model for Software in 1993 (Thayer et al, 1997).

As a result, requirement analysis emerges as a fast-growing development process in software engineering industry and the study of it becomes an important part to successful software development.

3.1.1 Requirements

To perform requirement analysis, one must first understand what a requirement actually is. IEEE defines a software requirement as the following (IEEE, 1983):

(1) A software capability needed by the user to solve a problem or achieve an objective.
(2) A software capability that must be met or possessed by a system or system component to satisfy a contract, specification, standard, or other formally imposed document.

3.1.2 Requirement Analysts

Requirement analysts are persons responsible for producing requirements in the requirement analysis process. Their main duties include identifying requirements from information gathered from different sources, structuring the information, and communicating the requirements to different audiences.

Requirement analysts are not just collecting and organizing information for clients and the target software, they have to help clients and potential users to identify and uncover their needs if they do not know them clearly. And since there are various people with different background involved in the requirement analysis process, requirement analysts have to make sure the requirements are presented in a consistent form that is easily understandable to different audiences.

In producing the requirements, the objective of requirement analysts should always be to maximize the options available to software designers in the next phase of the software development process. They should just specify what the software should do without

deciding how it can do the tasks. Therefore, they should not limit the available options for software designers.

3.1.3 Software Requirement Specification

The software requirement specification is the final product or document of the requirement analysis process. The production of it is the final goal of the whole process.

Requirement specification has three major objectives. First, it is used to achieve agreement regarding the requirements between system developers, customers, and end-users. This is important because frequent reviews of the requirements are usually needed to produce the final document. Second, it is used to provide the basis for software design. It should describe the suitable amount of information for software designers to proceed with the software development process. Third, it is used to support verification and validation because it provides a reference point to check whether the software satisfies all the requirements (Brackett, 1990).

3.2 General Development Approach

Similar to software engineering process, requirement analysis process is also broken into a number of phases. It consists of five activities: requirement elicitation, requirement analysis, requirement specification, requirement verification, and requirement management (Thayer et al, 1997). The followings give a brief summary of each:

Requirement elicitation is the process through which clients and the software developers discover, review, and understand the clients' and users' needs and the constraints on the software and the development activities.

Requirement analysis is the process of analyzing clients' and users' needs to come up with a definition of software requirements.

Requirement specification is for the development of a document that clearly and precisely records the requirements of the software.

Requirement verification is used to ensure that the software requirement specification is in compliance with the system requirements, conforms to document standards, and is an adequate basis for the design phase.

Requirement management is about planning and controlling the requirement elicitation, specification, analysis, and verification activities.

3.2.1 Requirement Elicitation

Once the business plan is created in the information engineering process, the requirement analysis team tries to identify what properties the software should have to meet the business ideas in the plan. Requirement elicitation is the very first step in which the requirement analysts have to gather and understand the information from clients and customers. The process involves finding the corresponding facts about the software, validating the understanding of information gathered from different people, and communicating open issues and conflicts for resolution. The participation of the clients is very important in the requirement elicitation because their background knowledge and experience can play a very strong role in the process.

Elicitation from clients is the most straightforward way of acquiring requirements. There are a lot of ways to get the requirements. The commonly used techniques include introspection, questionnaires, interviews, and focus groups.

3.2.1.1 Introspection

Introspection is an examination of a person's own thoughts and feelings. In this method, requirement analysts ask themselves what they want if they are the users and what the requirements of the software should be. This method can be very useful but it has the problem that introspection of software developers or requirement analysts is unlikely to reflect the needs of actual users. Therefore, the results of introspection can be inaccurate and its use can be limited. There should be other methods available to check the validity of the results from introspection (Thayer et al, 1997).

3.2.1.2 Questionnaires

Questionnaires are easily and widely used because they use statistical analysis and give scientific results. It is a neutral measurement method in which no human interaction is required. Therefore, no conversation can take place and human intervention during interviews that may affect the results can be avoided. However, the problem is that questions on the questionnaires must be relevant and can be interpreted easily and correctly without variation. Otherwise, valid and useful responses cannot be generated. Also, a large number of questionnaires is required to give meaningful results (Thayer et al, 1997).

3.2.1.3 Interviews

In interviews, the interviewer poses questions and then allows interviewee to answer them. The interviewer can get the answers and information from the interviewee through the interaction process in interviews. Any uncertainties on the questions can be solved immediately during the interviews. However, the main problem is that people do not

know how to describe a lot of things or unable to describe them clearly although they know how to do them (Thayer et al, 1997).

3.2.1.4 Focus Groups

Focus groups are group interviews in which groups are brought together to discuss topics with the requirement analysts together. More natural interactions between people are allowed and different opinions and information from different people can be received in a short period of time. However, the requirement analysts have to pick the people who can actually give constructive results, which is typically hard to predict and is a tough task. Also, some people may be unwilling to talk about certain subjects in front of others so the results may be inaccurate or incomplete (Thayer et al, 1997).

In fact, every method has some limitations. However, each method's strengths seem to complement the others so combinations of various methods can be usefully applied to various problems.

3.2.1.5 Common Problems

Even though many elicitation methods are available, eliciting requirements can usually be a difficult and troublesome task. The information from different resources may not be directly used by the analyst and it must be transformed to a usable form and see if the information is relevant. Also, clients may change their minds once they see or discover new possibilities more clearly. Therefore, the requirements are rarely static and changes are inevitable, and the elicitation process is an iterative process that feeds information to later processes in requirement analysis.

Although elicitation from users is a common way to acquiring relevant information for requirement analysis, it demands special skills of the requirement analyst due to the social techniques required for elicitation. However, most requirement analysts have technical background and lacks interaction and social skills. Therefore, it will take even more time for the time consuming process to complete.

Active client participation is a key factor to success in requirement elicitation as shown in Figure 3-1 (McConnell, 1996). However, the participation is often insufficient due to a number of reasons. First, clients do not exactly know what they want due to social, political, legal, financial or psychological factors. Also, clients may find it difficult to describe their knowledge about the problem because they often have limited technical skills. Therefore, different terminology may be used and can lead to misunderstandings. Because of all the above factors, clients may be unwilling to participate in the time consuming process and limited client participation always poses a general problem.

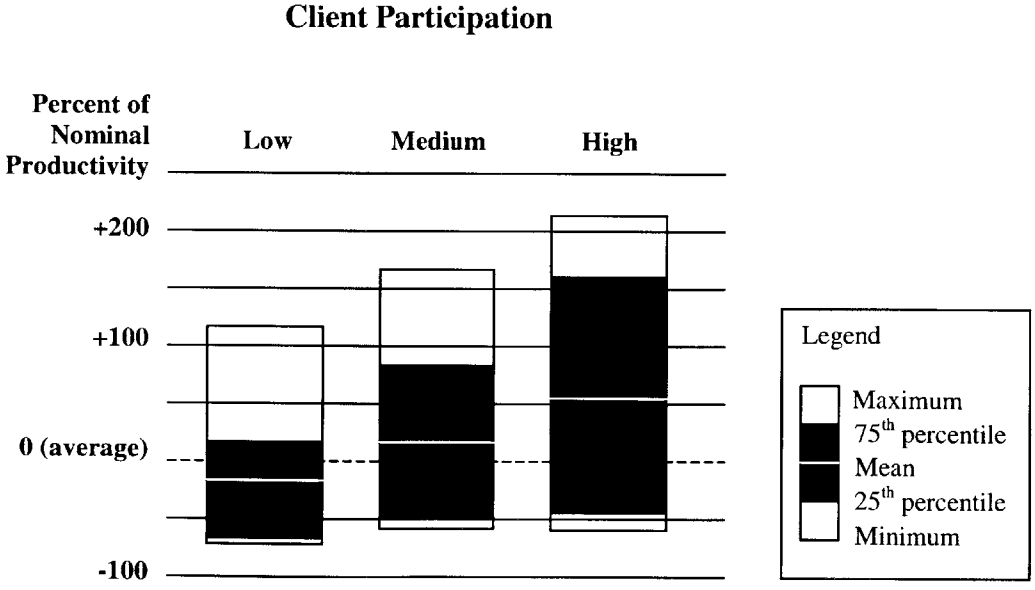


Figure 3-1: Effect of Client Participation on Productivity (McConnell, 1996)

As a result, the problems of requirement elicitation cannot be solved in a purely technological way, because social issues are much more crucial than in other requirement analysis processes.

3.2.2 Requirement Analysis

Now that the duties of requirement analysts are described, what a requirement specification is conveyed and methods of obtaining requirements are explained, the focus shifts to the actual requirement analysis. Requirement analysis is an information structuring process used to understand all the various parts of different requirements and their relationships. Each requirement analysis method has a different approach than other methods. However, the following five principles still hold for all different kinds of methods (Pressman, 1997):

1. All related information of the software must be understood and represented.
2. All functions of the software must be defined.
3. All behavior of the software must be represented.
4. All models used to must be partitioned to uncover details.
5. The requirement analysis process should move from essential information to implementation detail.

By applying these principles, the requirement analysts have a more organized way to do the analysis.

3.2.2.1 Structured Analysis

Over the years, many analysis and specification techniques have evolved. However, structured and object-oriented are the most widely used analysis methods.

Structured analysis evolved from functional decomposition and centered around functional requirements followed by the emergence of structured programming. It takes a distinct input-process-output view of requirements and uses processes to describe transformations of inputs to outputs. It draws on a common conceptual model for describing all kinds of problems and has procedures giving the direction of analysis and ordering of its steps. Also, it provides guidelines and heuristics about problem decisions and specifications. Moreover, it sets criteria for the evaluation of quality (Thayer et al, 1997).

The two most useful tools for structured analysis are data flow diagrams and data dictionaries. A data flow diagram provides graphic representation of data movement and transformations through the system while a data dictionary supports the data flow diagram by providing a repository for definitions and descriptions of each data item on the diagram (Thayer et al, 1997). They are easy to use because complex systems are hard to break down and visualize through textual representation and explanations.

Although structured analysis was very popular in the past, it still has a number of weaknesses. Two of the common critics about it are insufficient guidance provided by the conceptual model and little support for producing requirement specification. Since only weak constraints are imposed by the conceptual model and the model is relatively unconstrained, the quality of the requirement specification depends a lot on the experience of the requirement analysts. Also, structured analysis does not provide enough support for writing a high-quality requirement specification that can be easily read, referenced and reused (Thayer et al, 1997).

3.2.2.2 Object-Oriented Analysis

Object-oriented analysis differs from structured analysis in its approach to decompose a problem into parts and in its methods for describing the relationships between the parts. In object-oriented analysis, the requirement analysts decompose the problem into interacting objects based on the entities and relationships in the problem as opposed to functional decomposition in structured analysis (Thayer et al, 1997).

The primary focus in object-oriented analysis is on the entity rather than on the process. It centers on the underlying data that these processes are working on. The main focal point is the content of the entity, which is about the data structures of the entity, and the underlying states and aspect of the process (Thayer et al, 1997).

The objective of object-oriented analysis is to develop a series of models that satisfy the requirements for the software developed. The main intent is to define all the classes and the relationship and behavior associated with them for the software. The first thing is to identify all essential classes and define all the attributes. Then, the class hierarchy can be specified and essentials of the classes and attributes be analyzed. After that, the object-to-object relationship should be represented and the behavior of them be modeled. The whole process is repeated until the model is complete (Pressman, 1997).

The analytic principles of object-oriented analysis can provide many advantages. For example, the use of objects provides the means to divide the requirements into distinct parts and limit unnecessary dependencies between them. They also provide a basis for constructing reusable requirements. Object interfaces can be used to hide irrelevant details and provide only the essential information. Therefore, it reduces management complexity and improves readability (Thayer et al, 1997).

However, object-oriented analysis provides only informal specification and relies on design to add the details. The focus of object-oriented analysis is on problem analysis rather than specification and there is no formal evaluation to determine whether the later specification is complete or not. It simply assumes good specification will be produced if the analysis procedures are applied correctly. Also, analysts who use object-oriented analysis find difficulties in choosing appropriate objects and relationships just as the difficulties in choosing appropriate data flows and transformations in structured analysis (Thayer et al, 1997).

Despite all these problems, object-oriented analysis has become the more popular and dominant analysis method for requirement analysis process over the past few years.

3.2.2.3 Use Cases

After the requirement analysts gather all the requirements for analysis, they can create sequences of steps describing the interactions between a user and a system called scenarios to help identify the system to be constructed. In the past, requirement analysts used these scenarios informally and did not have a standard to govern their use. As a result, use cases came out as a documented way of linking scenarios by a common goal (Fowler et al, 1999).

Use case diagrams are now part of the Unified Modeling Language (UML). UML appeared after the emergence of object-oriented analysis and design. It is a standard modeling language for specifying, visualizing, constructing, and documenting the artifacts of software systems by simplifying the complex process of requirement representation and design using different kinds of diagrammatic representations. The rapid adoption of UML in software industry illustrates that modeling is an essential part for software development.

The success of use case has a number of reasons. It is a simple to use and understand and it can provide clear scenarios of interaction between the software and an actor, which is a role that a user plays in the system. Also, it provides a basis as graphical representations for software design and testing.

To create a use case, different actors of the system must first be identified. Then, different links between actors and use cases can be joined together to describe how the actor interacts with the system. Some typical representations of use cases include functions performed by an actor or system information acquired by the actor. Besides the simple links between actors and use cases, there are three kinds of other relationships between different use cases known as include, generalization and extend. Include is used to avoid repetition when there is similar description of certain behavior. It shows that one use case is included in another. Generalization is used for alternative scenarios when there is a small variation of a use case on another. Lastly, extend is used to describe a variation in a more controlled form. It requires the declaration of extension points in the base use case and the additional behavior of the extending use case can only be added at those extension points. The general use case template use case is shown in Figure 3-2. (Fowler et al, 1999).

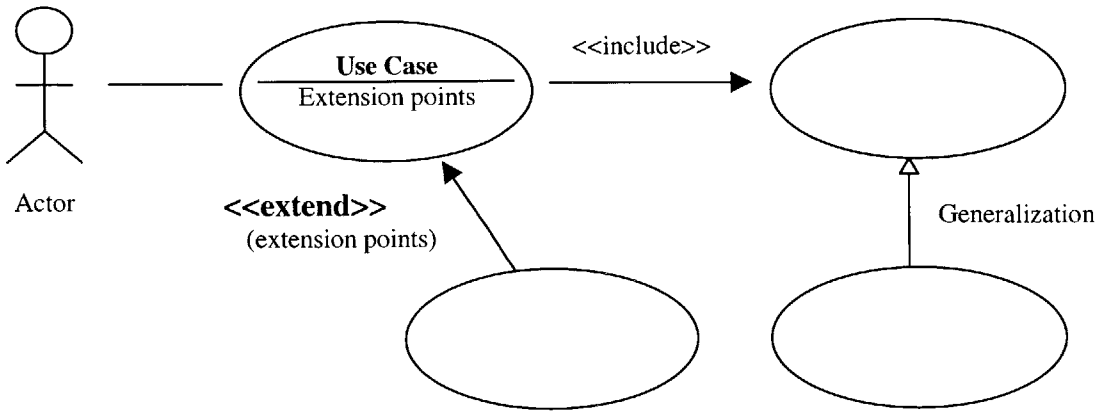


Figure 3-2: Use Case Diagram Template (Fowler et al, 1999)

3.2.3 Requirement Specification

Requirement specification can be viewed as a representation process after the analysis process. The success of the requirement analysis phase depends on how well the requirement specification is done. The effectiveness of the final product formed by requirement analysis process, the Software Requirement Specification (SRS), can directly affect the progress in later phases of the whole software development process. Any changes in the requirement that cause the SRS to be rewritten can significantly lower the productivity of the requirement analysis process as seen in Figure 3-3 (McConnell, 1996).

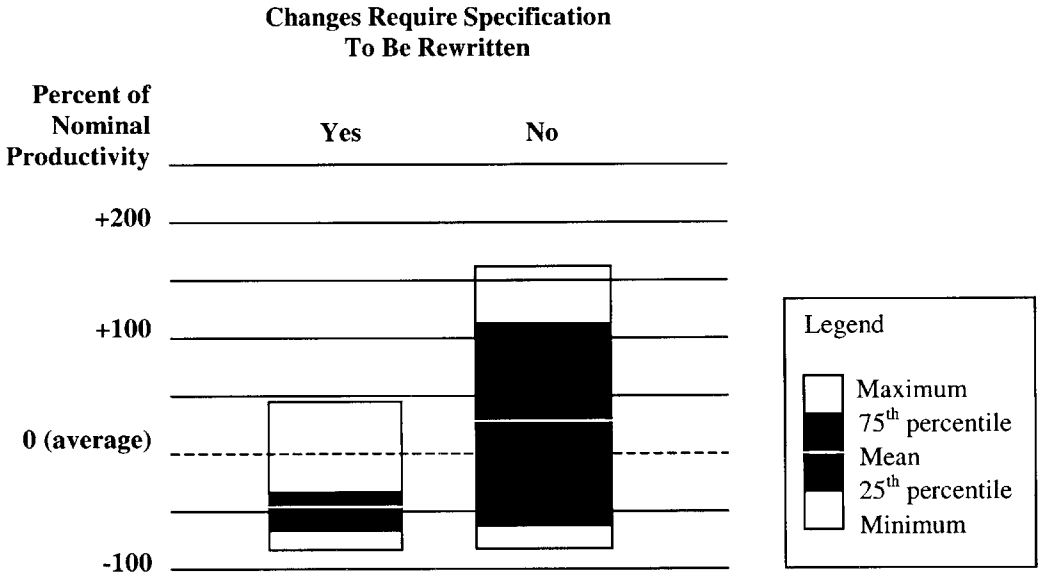


Figure 3-3: Effect of Rewriting SRS on Productivity (McConnell, 1996)

To ensure the quality of the SRS and to avoid it to be rewritten, standard procedures should be followed. The most widely used reference standard for producing software requirement specification is the IEEE Recommended Practice for Software Requirements Specifications. It describes the necessary content and qualities of a good SRS and

presents several samples SRS outlines. It is aimed at helping organizations to develop requirements for both in-house and commercial software products (IEEE, 1998).

This section, Requirement Specification, will address why SRS is used, its characteristics, and the linkage between SRS and a successful requirement analysis.

3.2.3.1 Usage

The SRS can provide a variety of benefits for customers and developers, some of them are listed as follows (IEEE, 1998):

- The SRS establishes the agreement between customers and developers on what software is to be built. It can be used later during the software development process as a basis to judge whether the software requirements are fulfilled.
- The SRS serves as one of the first standard documents produced in a software development process. By carefully preparing the requirements, omissions or misunderstandings can be minimized in later phases to avoid wasting development effort. It also helps to facilitate the idea transfer between customers and developers during the early software development process.
- The SRS provides a basis for estimating costs and schedules for the software project. The project managers can use it as the first document to track the development progress.
- The SRS provides a basis for software validation and verification. It can serve as a reference for software testing to determine whether the requirements are met and fulfilled.

3.2.3.2 Characteristics

However, a good SRS has to be produced in order to provide the above advantages. A number of characteristics of a good SRS should be learnt and practiced (IEEE, 1998):

- Correctness: the SRS should reflect the actual needs of customers and describe the exact requirements that the software should meet.
- Unambiguity: every terms and requirements in the SRS should only have one interpretation. Further explanations should be included to avoid future misunderstanding if there is any possible disagreement.
- Completeness: the SRS should contain all the information needed for later software development phases in a standard form. Any omission or non-standardized format is considered as incompletion of SRS.
- Consistency: the requirements in SRS should not conflict with one another. Standard definitions and terminology should be used to avoid any possible conflicts.
- Ranked for importance: the rankings will help software developers to decide how they should spend the amount of time and effort to different requirements.
- Verifiable: the SRS and the requirements are verifiable if it is possible to determine whether the outcome represents what the requirements specify. Usually, the requirements should be stated in concrete terms and measurable quantities.
- Modifiable: the SRS should be organized for ease of change with same structure and style through the document. Requirements should be written separately and not be redundant.
- Traceable: each requirement in SRS should have a clear origin and is easy for reference in future development process.

Some simple templates are given by IEEE for the SRS with different organizations of requirements for different needs. Each of them contains a simple outline and a good SRS

should include the information as suggested in the IEEE standard. A sample template is shown in Figure 3-4 (IEEE, 1998).

The sample outline is just a general guideline. Based on the nature and size of each project, some sections may not be needed and can be omitted. The whole idea is to provide requirement analysts an aid to produce a qualified SRS.

3.2.4 Requirement Verification and Validation

As with any process, the requirement analysis process in the project needs to be verified for completion and accuracy. Requirement verification checks that the allocation of system requirements to software is appropriate and correct. It consists in showing that the requirement specification is comprehensible, precise, complete and consistent. It is used to check whether the requirement analysts are writing the requirements right (Thayer et al, 1997).

Requirement validation determines the correctness of the requirement specification with respect to the users' needs and requirements. It consists in confirming with the customer that the requirements are correctly understood and documented. It is used to check whether the requirement analysts are writing the right requirements (Thayer et al, 1997).

Table of Contents

Figure 3-4: Sample Template of SRS with Section 3 organized by Feature (IEEE, 1998)

3.2.4.1 Common Errors

During the requirement analysis and specification processes, there may be errors which can lead to the failure of subsequent activities. There are four common types of errors in the SRS: omission, incorrect fact, inconsistency, and ambiguity. On average, over 250 errors are detected, and the percentages of different types of errors present in a typical SRS are shown in Table 3-1 (Jalote, 1997).

Table 3-1: Percentage of Different Errors in Requirement Specification (Jalote, 1997)

Omission	Incorrect Fact	Inconsistency	Ambiguity
26%	10%	38%	26%

Since the requirement specification is used as the basis for subsequent software development, it is essential that it is verified and validated. Requirement verification and validation is a unified approach to identify and resolve problems early in the software development process. It employs a method for evaluating the correctness and quality of the requirements.

3.2.4.2 Method

Requirement reviews are the most common method of requirement verification and validation. Both software developer and customer conduct the review of the SRS. During the review, each person looks for errors or other matters of concern in the SRS. There are many ways to conduct a review. One of them is to let the participants read through the SRS first and then have the requirement analysts answer any questions they may have. Another way is to have the requirement analysts explain the SRS first, and then the participants can ask their questions. In either method, the discussions between

different participants during the review form the most important part of the process. The focus is on group dynamics instead of giving comments or questions individually (Thayer et al, 1997).

Some of the important software verification and validation processes are project concept checking, software traceability analysis, and software requirement evaluation. Project concept checking determines whether the SRS satisfies the project objectives in terms of the software performance and requirements. Software traceability analysis traces the SRS to system requirements and checks the relationships for consistency and correctness. Lastly, software requirement evaluation checks whether SRS fulfills the standard requirements (Thayer et al, 1997).

Once the review is complete, the SRS is completed and the document will be frozen. Then, it will be signed by both the customer and the developer and it becomes a legal document for software development, which is yet another reason for thorough verification and completion of the SRS. Changes in requirements after the specification is finalized will result in a change of project scope and therefore can increase cost and/or cause delays in the project schedule.

Even with the best review procedures in place, a number of common specification problems persist. The SRS is difficult to verify and validate in a complete way, so further testing procedures are still needed in the later software development processes.

3.2.5 Requirement Management

A survey showed that the top three reasons of project failure all had to do with poor requirement management practice: lack of user input, incomplete requirements, and frequently changing requirements. Another survey of projects found that more than half

the projects suffered from inadequate requirement management (McConnell, 1996). Both of them presented results that showed requirement management is a high-risk process which can lead to project failure. As a result, requirement management should be done properly to avoid any potential project failure.

3.2.5.1 Purpose

Requirement management involves the activities and tasks undertaken by one or more persons for the purpose of planning and controlling the requirement analysis activities in order to achieve objectives that cannot be achieved by the others acting alone. It also allocates the resources for the requirement analysis team such as budget, time, and personnel.

Generally, proper requirement management can improve development speed in two ways. First, requirement management can speed up the requirement elicitation process. Since people involvement is critical in elicitation, requirement management should consistently give a sense of urgency during the process to help prevent the excessive resources spent on it. Second, it usually costs 50 to 200 times less to get a requirement right at the first place than fixing the requirement later. Therefore, good requirement management practices can help to reduce the cost of requirement changes by controlling the requirement analysis and specification processes (McConnell, 1996).

3.2.5.2 Key Process Area in Capability Maturity Model

Because requirement management is a critical and important process, the Software Engineering Institute designated it as one of the key process areas in the Capability Maturity Model. Each maturity level in the CMM, besides Level 1, is decomposed into

several key process areas that identify issues which an organization should focus on to achieve a maturity level. The key process areas at Level 2 focus on the software project's concerns related to establishing basic project management controls. One of the key process areas in Level 2 is Requirement Management which describes how requirement analysis should be done to achieve the maturity level (Paulk and Weber et al, 1993).

The purpose of Requirement Management is to establish a common understanding between the customer and the software project of the customer's requirements that will be addressed by the software project. This agreement with the customer is the basis for planning and managing the software project. There are two main goals for Requirement Management:

1. System requirements for software development are controlled to establish a baseline for management.
2. Software plans, products, and activities are kept consistent with the system requirements for software development (Paulk and Weber et al, 1993).

To achieve these two goals, the software engineers need to control the relationship with the customer by adopting an effective change control process. Also, they have to review the initial and revised system requirements and ensure that the requirements are well documented and controlled.

3.2.5.3 Suggested Practices

There are some useful suggestions based on practical experiences to improve the requirement analysis process and ensure that it is successful and productive (Thayer et al, 1997).

Getting the clients and developers involved:

Lack of communication between the clients and developers is the most common problem of requirement analysis, especially during the requirement elicitation process when substantial client involvement is needed. If the client does not voice his or her needs, the software developers will structure the project according to their own views of what the client wants. This can result in unhappy clients. Another scenario is that long delays in the project will materialize when the client does finally express his or her requirement needs and the developing team will then have to backtrack and perhaps start over. Therefore, one of the main tasks of requirement management is to get the clients and developers together early in the process to solve any problems they may have.

Getting the needed information and requirements:

Clients usually have difficulties to identify what they actually want from the software or they even do not know exactly what they want. Therefore, one concern of requirement management is to help clients to broaden their perspectives and to access useful information that may be hidden in the environment or in the minds of the users.

Although clients sometimes find no difficulties to express what they want, requirement changes usually happen during the requirement analysis process. Therefore, another concern of requirement management is to keep track the requirement changes and allocate enough time for changes.

Assure the use of standard notation and format:

The use of standard notation and format should be monitored by requirement management throughout the requirement analysis process. This is especially important in

the distributed environment, where the developers rely on notations more heavily due to the absence of face-to-face communication. If notations are done properly, the time and other resources for error checking and requirement verification and validation will significantly reduce.

Communicate with other development teams:

Besides helping to facilitate the communication within the requirement analysis team, another purpose of requirement management is to communicate with other development teams such as project management team and make sure the requirement analysis team get updated with the project progress and other teams' work.

3.2.6 Challenges

Although the requirement analysis process has five different complicated activities, many software developers often ignore the importance of it and the challenges they face in the process. As a result, the resources allocated to it is very limited compared to other phases in software development as seen in Table 3-2 (Jalote, 1997).

Table 3-2: Effort Distribution in Software Development (Jalote, 1997)

Requirements	10%
Design	20%
Coding	20%
Testing	50%

The insufficient effort in requirement analysis can often lead to many errors in requirements. However, the cost of fixing them is relatively low compared to other development phases as seen in Table 3-3 (Jalote, 1997).

Table 3-3: Cost of Fixing Errors in Software Development (Jalote, 1997)

Phase	Cost (person-hours)
Requirements	2
Design	5
Coding	15
Acceptance Test	50
Operation and Maintenance	150

Therefore, the proper practice of requirement analysis should be observed by the software developers to overcome any challenges they face in this process. More resources should be allocated to it and complete error checking should be done here to minimize the cost of fixing the errors later in the development process.

3.3 Distributed Development Approach

The requirement analysis process is already difficult for co-located teams to coordinate the group work. However, it is even more challenging for geographically distributed teams. When team members are in different locations, the opportunity for direct face-to-face collaboration may be non-existent. Although improvements in various communication tools, such as high speed modem lines and videoconferencing, have significantly eased the collaboration problems for distributed teams, the current simple communication media alone may not be sufficient enough to support the intense and frequent collaboration between co-located teams during the requirement analysis process (Steinfield et al, 1999).

3.3.1 Specific Needs

Because of the geographically distributed nature of this development environment, specific needs for human contacts and ways of handling information are required to make the requirement analysis process a successful one.

There is a potential danger for distributed development environment because each team usually tends to coordinate its own activities and tries to minimize the need for interactions with other teams. Distributed teams usually divide the tasks in a way that makes collaboration minimal. Since distributed teams consist of members from different locations and organizations, they work in different environments with various social and cultural contexts. These factors often contribute to many communication and coordination problems, especially in requirement analysis process in which substantial communication and collaboration is required in every step between different people. Research shows that there are different needs that arise from distributed teams for the success of collaboration between them in the requirement analysis process (Steinfield et al, 1999).

3.3.1.1 Contacts

There are four modes of contacts for the requirement analysis process: asynchronous collaboration, synchronous collaboration, informal contacts, and tele-presence. All of them are important for the collaboration between distributed team members.

Asynchronous collaboration:

Asynchronous collaboration is a way of collaboration between people occurring at different times such as electronic mails. Therefore, distributed team members can

generate requirements and access all the relevant documents individually at different times using asynchronous collaboration. It is particularly important for distributed teams located in different time zones when synchronous meetings are hard to schedule. It is also useful for setting up meetings and group coordination, and certain issues that are not urgent and take longer time for each member to think and analyze independently (Herlea et al, 1998) (Steinfield et al, 1999).

Synchronous collaboration:

Synchronous collaboration is a way of collaboration between people occurring at the same time such as phone calls. Real time collaboration and discussions are still essential for various activities in the requirement analysis process to help clarifying and validating any issues that different team members may have despite the use of asynchronous collaboration. Any conflicts or problems can be solved quickly without any further misunderstandings. It is particularly important for rapid development of software and when the project is under time pressure. However, even though such resources for synchronous communication are available, they may not be easily accessible due to scarcity and unpopularity (Herlea et al, 1998) (Steinfield et al, 1999).

Informal contacts:

Informal contacts are forms of unscheduled collaboration such as chatting with other people during lunch hours, as opposed to any pre-scheduled meetings. Spontaneous and informal real-time interaction activities should be frequently held for improving distributed teams' performance. Distributed team members can get to know each other better through casual contacts and this will enhance the cooperation and communication processes between them. Although the importance of these activities has well been

recognized, the opportunities for these activities are particularly limited or even non-existent for distributed teams due to time and location differences (Steinfield et al, 1999).

Tele-presence:

Tele-presence means the sense of presence through the use of technology although people are physically in different location. During the requirement analysis process, it is necessary to give distributed team members the sense of presence that they are meeting together in the same place. It helps to provide an environment for group discussion, negotiation and decision making processes (Herlea et al, 1998).

3.3.1.2 Information

Other than the essential collaboration contacts, information storage for distributed team members is also important for requirement analysis.

Tele-data:

Tele-data is the way to share data through the use of technology for people in different locations. Throughout the requirement analysis process, huge amount of documents are generated by distributed team members. The ability to share these various forms of data for later access is essential from brainstorming to finalized standard specifications. Therefore, distributed teams need to have a common place to share information and documents, or any relevant information needed to be shared between distributed team members. It should be stored in such a way that the information is easy to store and retrieve (Herlea et al, 1998).

Progress report:

It is important for any project team to have methods of checking themselves to keep things progressing and within the project schedule. This is especially so for distributed teams. Distributed team members cannot get immediate update on project progress because they cannot directly and easily communicate with each other as often as co-located team members. Therefore, each distributed team member should explicitly report his or her progress regularly in formal procedures to avoid confusion. Otherwise, each team member may work on a part that someone else has already completed, or even think that others are inactive which can ultimately decrease the team spirit and morale (Steinfield et al, 1999).

3.3.1.3 Others

Besides the contacts and information issues, there is also one concern on the technology side. Different distributed team locations may often have heterogeneous or even incompatible technology infrastructures. Different computer operating systems or software's are the usual problems and distributed team members may need to take extra effort in order to keep up the normal communication and document sharing activities (Steinfield et al, 1999).

3.3.2 Required Tools

Based on the specific needs of the distributed requirement analysis process, there are some required tools that are needed to satisfy those needs. However, the software engineering industry is currently placing more emphasis on technical issues than social problems faced by geographically distributed teams. It is considered that the two aspects of team support, technological and social, are not covered in equal measure. The

technical improvement shows significant increase in productivity through better development methods, faster machines and tools. However, the lack of focus on improving the tools for social issues can lead to serious problems because human interactions are very important in the requirement analysis process (Haag et al, 1997). Little attention has been given to how advanced tools can support collaboration specifically for requirement analysis process although different tools for distributed collaboration are already studied widely.

In this section, the reasons for development of these tools specifically for the requirement analysis process are discussed first. After that, some fundamental features of them are presented.

3.3.2.1 Developmental Reasons

The complicated requirement analysis process becomes a more difficult task when clients and developers are located in different locations. One possible solution to this problem is to develop a groupware, one kind of technology designed to facilitate the work of groups and collaboration between people, specifically for the requirement analysis process.

Research shows that requirement analysis is one of the areas of new product development where the deployment of groupware solutions can lead to significant process improvements. There are several reasons why requirement analysis process is more amenable to groupware support than other phases of software development. During the requirement elicitation process, there is a strong need to have good communication and collaboration methods and procedures between different people. Also, during the requirement analysis phase, it is usually cheaper and easier to shorten the software development process by correcting the errors and mistakes. Many researches indicate

that problems occurred later in the software development processes are caused by poor or conflicting requirements. Therefore, groupware for requirement analysis process would definitely help to save the cost and time for distributed requirement analysis. Furthermore, the payoff of groupware support in the requirement analysis process would be higher because of the extensive communication needs of clients and developers during the early phase of the software development project (Salo et al, 1999).

3.3.2.2 Fundamental Features

The fundamental features of the required tools can be grouped to three areas based on the specific needs of the distributed requirement analysis process. The first feature that the groupware should have is the meeting space that supports the requirement analysis process. The second one is the ability to express and negotiate multiple perspectives on requirements. And the last one is the function as a web repository for storing requirements and other information.

Creating a meeting space that supports the requirement analysis process:

The groupware should provide pre-customized meeting rooms and also allow customization of them by users in order to facilitate the special use of requirement analysis process. It should provide a main diagram showing the whole requirement analysis process from requirement elicitation to requirement verification and validation. Each of these steps should include several other activities, and the groupware should also provide particular settings and tools for each of these activities to enhance the interactions between distributed team members. The collection of the main diagram and meeting rooms provides a guide for the requirement analysts to follow the practical requirement analysis procedure. However, it should also offer the flexibility that users can choose not to follow these steps and creates their own steps (Herlea et al, 1998).

By using this meeting room feature, different participants in the requirement analysis process can collaborate synchronously. They can also use this as informal contacts with each other.

Expressing and negotiating multiple perspectives on requirements:

During the requirement analysis process, distributed team members usually have significant differences in perspectives on different issues. They usually have different background and use different terminology. Their levels of knowledge and their organizational considerations and cultures may also be very different. Since they have no prior interactions before, it will be especially difficult for them to resolve these differences.

The groupware should support the requirement analysis process by having concept mapping tools. With these tools, clients and developers can use the representation schema in the concept map tools to structure their opinions and viewpoints. All participants can discuss and negotiate together in real time on different perspectives. They can simultaneously modify the requirements and see the changes so any misinterpretations can be resolved immediately. This will give the participants a virtual feeling of discussing topics together in the same room (Herlea et al, 1998).

Using a web repository for storing requirements and other information:

This feature supports the collaborative nature of requirement analysis using the web interactive technologies and allows the distributed team members to store documents. It can be used to identify, elaborate and organize requirements for requirement specification. Also, it should provide a logical mechanism for identifying, organizing

and justifying the requirements. In addition, it should also show the relationships between certain relationships and allow traceability by storing them in a hierarchical way. Furthermore, it should establish a system for storing requirements in the web repository for easy access and retrieval for all the distributed team members. Distributed members can also use this as a way for asynchronous communication to setup meetings and submit progress reports (Anton et al, 1996).

3.3.3 Importance

Through the discussions on distributed requirement analysis, one can see that it is just a normal requirement analysis process in software engineering with certain restrictions. The most critical restriction is the geographically distributed teams in different locations. By imposing this restriction, collaboration between distributed teams becomes more difficult and the urgent need for tools with features specifically for requirement analysis to enhance collaboration is no longer questionable. A lot of research has been done on this area and the importance of distributed development approach of requirement analysis is not debatable anymore.

3.4 Summary

After the in-depth discussion on the process of requirement analysis, it can be proved that Dr. Brooks' renowned view on the difficulties about requirement analysis back in 1987, as quoted in the beginning of this chapter, is still applicable today. The challenge of distributed development further complicates the process of requirement analysis. As a result, collaboration between different participants in the requirement analysis process should be improved through the use of advanced technology. Then, distributed

requirement analysis – the first step towards a successful distributed software development project, will longer be a difficult task in the near future.

Chapter Four

Case Study:
Distributed Software Engineering Project – ieCollab

ieCollab, which stands for **i**ntelligent **e**lectronic **collab**oration, was created so that graduate students from three schools located in three different countries could work on a distributed software engineering project. The basic software development procedures and techniques were studied throughout the project. More importantly, the specific problems anticipated by students with regard to collaboration, coordination, and cooperation in the distributed development environment were learned through their experience in the project. Therefore, the different distinctive features of ieCollab provide the invaluable elements for the study of distributed software development.

In this chapter, the project and distributed teams of ieCollab are first introduced, followed by the discussion of requirement analysis process in ieCollab. Then, the problems on the development of ieCollab and recommendations on how to improve distributed software development process based on the ieCollab experience are presented.

4.1 Project Description

ieCollab has a unique project background with special objectives, and at the same time, it has certain common characteristics from different versions as in various distributed software development projects.

4.1.1 Background

ieCollab is a nine-month software development project that began in September 1999 and ended May 2000. The development team consisted of three advisors and thirty-four students from three different universities: the Massachusetts Institute of Technology (MIT) in the United States, Centro de Investigación Científica y de Educación Superior de Ensenada (CICESE) in Mexico, and Pontificia Universidad Católica (PUC) in Chile.

Students participated in the project as a class named "Distributed Development of Collaborative Engineering Support Systems". The course covers the technical and organizational principles, methods and tools required for collaboration in successful distributed software development. The team project involves improving Internet-based communication tools to enable synchronous and asynchronous collaboration as well as casual contact and social interactions. Various development tools and practices such as the Unified Modeling Language (UML), the Java programming language, and the IEEE Software Engineering Standards were used extensively for the project.

The basic idea behind ieCollab is to extend the **C**ollaborative **A**gent **I**nteraction and Synch**ro**nization (CAIRO) software project, which was initiated at MIT in 1995. CAIRO is a platform-independent application enabling geographically distributed collaboration in a virtual environment and it was first introduced to the class as a software engineering project in 1997.

4.1.2 Objectives

ieCollab is an internet-based collaborative application service provider for communicating information and sharing software applications in a protocol-rich Java meeting environment. It is created to help different organizations manage distributed teams by providing an effective tool to solve collaboration problems in order to reduce project costs and duration (Mills, 2000).

Through the development of ieCollab, students would learn the software development process from the start of gathering business ideas to the end of delivering the final product and gain knowledge in solving problems while challenged by a time line. In addition, students experimented the problems they faced and the mistakes they made. By learning from these difficulties, students should be able to develop the skills to cope with different situations arising over the course of the project and could provide recommendations to solve them.

4.1.3 Scope

For any project, a goal or end product must be established. The team's original vision of ieCollab was to launch four versions. A base version was to be created first, and the rest would be upgrades. Each version would focus on a specific functional component of the overall ieCollab system. The four versions are identified in Table 4-1.

Table 4-1: Four Versions of ieCollab

Number	Name	Descriptions
1	Meeting Management	Allows distributed users to setup and manage online meetings and includes basic meeting functionality such as the use of meeting agenda.
2	Transaction Management	Allows the ieCollab server to keep track of users' transaction and usage of ieCollab's meeting management services.
3	Collaboration Server	Allows the use of different interactive collaboration tools such as chat tools for distributed group communication.
4	Application Server	Allows the use of distributed users to work on the same documents concurrently using third-party applications.

Soon after the formulation of the described project goal, while still at the beginning of the project, the whole development team decided to develop the first two versions of ieCollab as the project scope and would continue to develop the last two versions when time was available.

However, due to the delays throughout the development process and the departure of a third of the original team, the project scope was changed to the development of working programs of the two versions only. At the end of the project, only the compiled code packages were delivered for the two versions of ieCollab.

4.2 Project Organization

Each software development project has its own development process, approach, and environment for its specific needs. ieCollab developers used the most common and recent way to organize for the efficient and effective development for this project.

4.2.1 Process

The ieCollab project used the traditional waterfall model during the early phase of the software development process. Each development phase was completed one by one and the next phase did not start until the end of the previous phase. This made the project planning easier for the project management team.

However, due to substantial delays in the early phase of the project, incremental model was used later. It allows more flexibility so that the next phase of the development process can start without waiting until the end of the previous one. Therefore, different team members got involved earlier and worked on the project without the wait for the completion of the previous phases.

4.2.2 Approach

Throughout the entire ieCollab project, the object-oriented approach was used. It allows the easy transition between the different phases. Also, the object-oriented programming language Java was used for programming team which allows the improvement on code size and efficiency than structured programming languages. In addition, this approach is easy to extend and reuse of the work in different development phases which is particularly important for the later continual on the same project or development processes for similar project in the future.

4.2.3 Environment

The ieCollab project was developed in a distributed environment. Team members were geographically located in three different countries: the United States of America, Mexico,

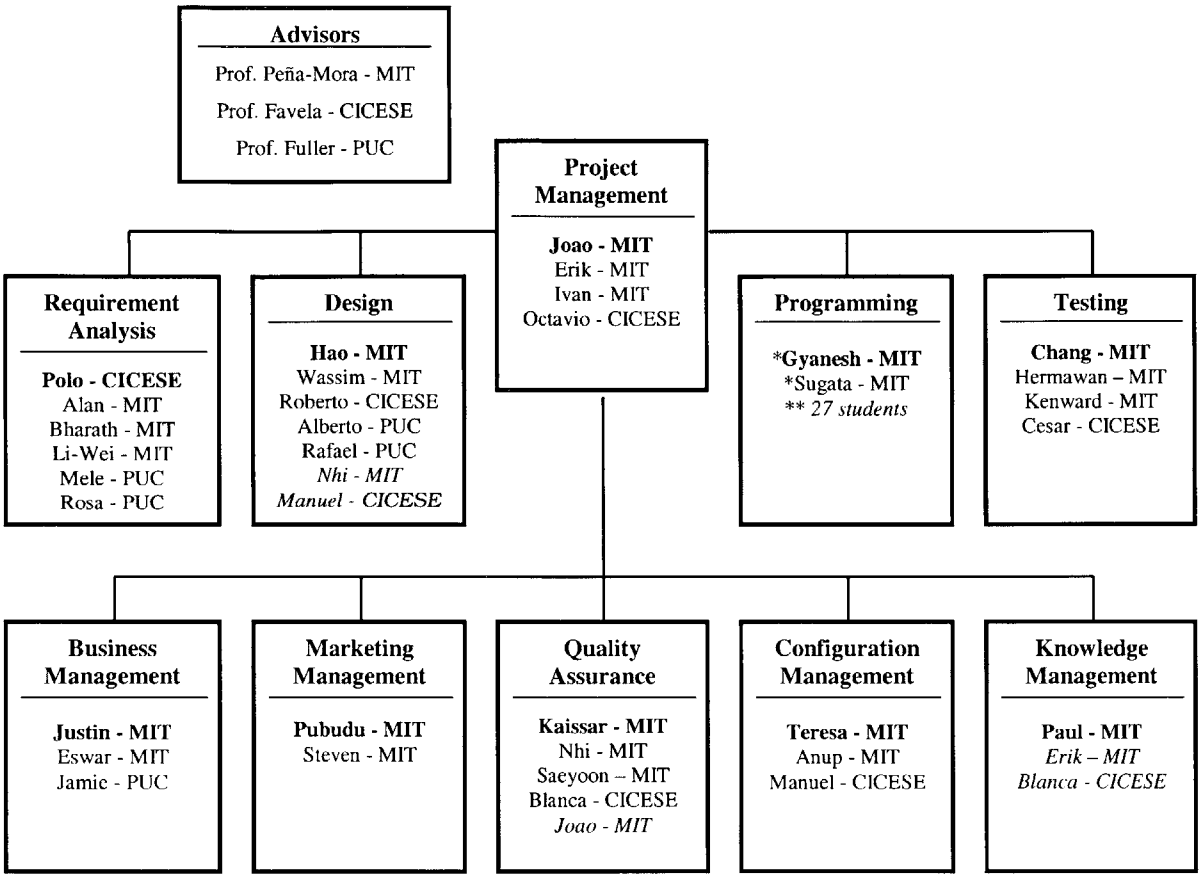
and Chile. The tools that were used to communicate among distributed teams included electronic mails, ICQ, and Microsoft NetMeeting. Phone calls and audio and video conferencing were sometimes used during the class periods to support distributed learning and collaboration.

4.3 Team Organization

The ieCollab team was organized in such a way to enhance the performance of each team and achieve the objectives of this project. Each team consisted of different numbers of members from different schools and students had the chance to choose their own roles according to their knowledge and interest on different roles. Each team had its own tasks and goals and all the teams coordinated their work internally as well as cooperated externally with other teams to achieve the project goals and tasks.

4.3.1 ieCollab Team

There was a total of ten teams in the ieCollab project. The ieCollab team consisted of thirty-four members. Twenty-three of them are from MIT, six of them are from CICESE, and five of them are from PUC. Each member had first and secondary roles in the project except the two programming team members who needed to focus on the heavy workload of programming. Because there was a great deal of programming and it was so time consuming, only five members did not have programming support as their secondary role. Besides, the three professors from the three schools act as advisors by providing suggestions on the market and areas in which the development team should focus for this project. The initial team organization figure and team member distribution table are shown in Figure 4-1 and Table 4-2 respectively.



Notes:

* No 2nd role assigned to these members
** Members who have no other 2nd roles

Legend:

Team Name
Leader - School
Member (1st Role) - School
Member (2nd Role) - School

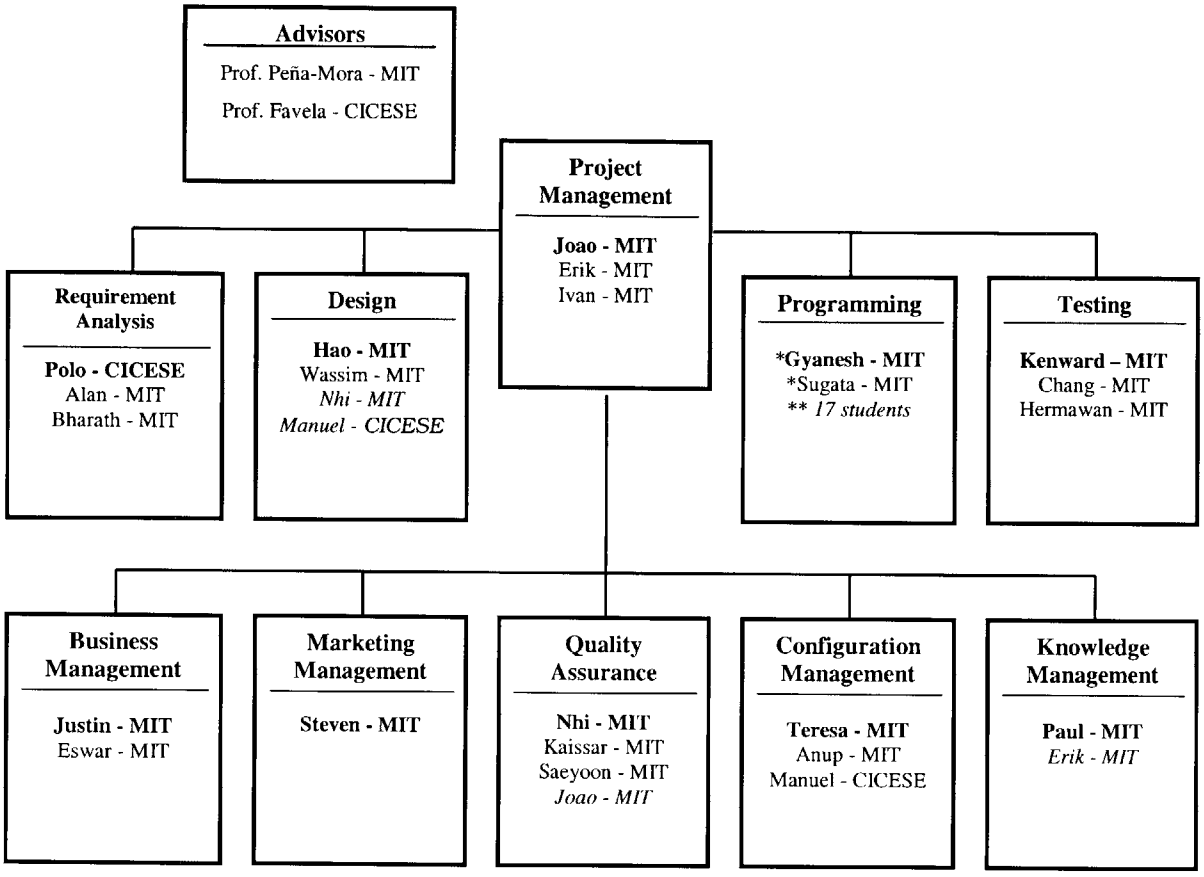
Figure 4-1: ieCollab Initial Team Organization

Table 4-2: Initial ieCollab Team Member Distribution

Team	Number of Members: 1st Role (2nd Role)			
	MIT	CICESE	PUC	Total
Project Management	3	1	0	4
Business Management	2	0	1	3
Marketing Management	2	0	0	2
Requirement Analysis	3	1	2	6
Design	2 (1)	1 (1)	2	5 (2)
Programming	2* (18)	0 (4)	0 (5)	2 (27)
Testing	3	1	0	4
Quality Assurance	3 (1)	1	0	4 (1)
Configuration Management	2	1	0	3
Knowledge Management	1 (1)	0 (1)	0	1 (2)
Total	**23 (21)**	**6 (6)**	**5 (5)**	**34 (32)**

Notes: * No 2nd role assigned to these members

After the first three months of the project, there was a significant change in the ieCollab team organization. A total of eleven members, which is about one-third of the entire team, left the project. The entire group from PUC, five members, left, four others were from CICESE, and the remaining two were from MIT. As a result, leadership changes occurred for certain teams. The final team organization figure and team member distribution table are shown in Figure 4-2 and Table 4-3 respectively.



Notes:

* No 2nd role assigned to these members
** Members who have no other 2nd roles

Legend:

Team Name
Leader - School
Member (1st Role) - School
Member (2nd Role) - School

Figure 4-2: ieCollab Final Team Organization

Table 4-3: Final ieCollab Team Member Distribution

Team	Number of Members: 1st Role (2nd Role)			
	MIT	CICESE	PUC	Total
Project Management	3	0	0	3
Business Management	2	0	0	2
Marketing Management	1	0	0	1
Requirement Analysis	2	1	0	3
Design	2 (1)	0 (1)	0	2 (2)
Programming	2* (16)	0 (1)	0	2 (17)
Testing	3	0	0	3
Quality Assurance	3 (1)	0	0	3 (1)
Configuration Management	2	1	0	3
Knowledge Management	1 (1)	0	0	1 (1)
Total	**21 (19)**	**2 (2)**	**0**	**23 (21)**

Notes: * No 2nd role assigned to these members

4.3.2 Team Tasks

Each ieCollab team had its own specific tasks to be accomplished. After certain phases of the project, each team needed to submit certain deliverables to mark the finish of that phase and the completion of all the tasks. The following details the tasks of each of them:

Business Management Team:

The business managers were responsible for defining the goals of ieCollab by studying the needs of customers in order to create competitive advantage for them through the product. They had to create a business plan for ieCollab at the beginning of the project.

Marketing Management Team:

The marketing managers were responsible for identifying the market needs and changes through extensive market research and deploying the sales strategies for the pricing schemes of ieCollab. They also worked closely with the business managers to create the business plan.

Project Management Team:

The project managers were responsible for the whole ieCollab project planning and organization. They needed to coordinate all the activities and maintain effective communication among different teams. They had to create a project management plan which specifies the tasks, schedule, and all the detailed management issues of the project.

Requirement Analysis Team:

The requirement analysts were responsible for translating the business ideas from business managers to software requirements. They needed to determine the features and define the main functions of ieCollab by capturing all the requirements. They had to create the software requirement specification for each version of ieCollab, which provided the reference for design and testing.

Design Team:

The designers were responsible for communicating the requirements from the requirement analysis team to the programming team by designing the functions of ieCollab and deciding how it works. They had to create the design specification for each version of ieCollab, which provided the basis for programming.

Programming Team:

The programmers were responsible for generating the executable codes of ieCollab. They needed to create concise, quality and documented codes using Java language that were easy to read and understand to help the testing procedures later. The source codes for each version of ieCollab were the deliverables for the programming team.

Testing Team:

The testers were responsible for ensuring the integrity of the software by finding any possible errors in the program. They developed different test cases in the test plan for different kinds of tests to be run for ieCollab. The testing specification for each version of ieCollab and the final testing report with testing results were the major items delivered by the testers.

Quality Assurance Team:

Quality assurance engineers were responsible for maintaining the standard practices and quality throughout the development process. They conducted inspections, walkthroughs and audits to ensure the quality of the documents was up to the IEEE standard that was used by ieCollab. The quality assurance plan was the main product delivered by the quality assurance engineers.

Configuration Management Team:

The configuration managers were responsible for controlling the versioning system and changes occurred after final approval of documents by using the Concurrent Versions System (CVS) and setting up of Change Control Board (CCB). In the CCB,

configuration managers needed to review all change requests and report any actions taken after the request. Besides the delivery of the configuration management plan, a repository web site (http://cee-ta.mit.edu/1.120/cm/index2.html) was setup to store all the approved documents.

Knowledge Management Team:

The knowledge managers were responsible for maintaining the information generated during the software development process. They needed to make sure all the documents were properly stored and were ready for retrieval and change through the development process. Other than the knowledge management plan, the knowledge managers had to produce the user and technical guides for ieCollab. Also, they had to maintain the ieCollab project repository web site (http://collaborate.mit.edu/1.120.html) which was used to store all the documents for the project.

All the deliverable items of different teams are summarized in Table 4.4.

Table 4-4: ieCollab Team Deliverables

Deliverable Item	Responsible Team
Project Management Plan v1.2	Project Management
Project Management Presentation	Project Management
Business Plan v1.0	Business Management Marketing Management
Requirement Specification Meeting Management v1.4	Requirement Analysis
Requirement Specification Transaction Management v1.6	Requirement Analysis
Requirement Analysis Presentation	Requirement Analysis
Design Specification Transaction Management v1.0	Design
Design Specification Meeting Management v0.5	Design
Design Specification Client Interface v0.3	Design
Design Presentation	Design
Source Code Meeting Management	Programming
Source Code Transaction Management	Programming
Programming Team Presentation	Programming
Testing Specification Meeting Management v2.0	Testing
Testing Specification Transaction Management v2.0	Testing
Testing Specification System and Integration v1.0	Testing
Testing Reports	Testing
Testing Presentation	Testing
Quality Assurance Plan v2.0	Quality Assurance
Quality Assurance Presentation	Quality Assurance
Configuration Management Plan v1.0	Configuration Management
Configuration Management Presentation	Configuration Management
Knowledge Management Plan v0.2	Knowledge Management
User Guide	Knowledge Management
Technical Guide	Knowledge Management
Knowledge Management Presentation	Knowledge Management

4.4 Requirement Analysis for ieCollab

The ieCollab requirement analysis team had some unique characteristics and differences than other teams in the context of team structure and goals. All these special distinctiveness affected directly the work accomplished by the requirement analysis team.

4.4.1 Requirement Analysis Team

The ieCollab requirement analysis team consisted of six members at the beginning of the project. The team leader was Leopoldo (Polo) Moran from CICESE. Three team members Bharath Krishnan, Li-Wei Lehman, and Yu Lung Alan Ng are from MIT, while the other two members Rosa Alarcon and Maria Elena Ruiz-Tagle are from PUC.

Two months after the project started, Li-Wei and the two members from PUC left the project. The remaining three members continued the work of requirement analysis team till the end of the project.

The ieCollab requirement analysis team was the only team besides design team that had members from all three different universities. Also, it was the only team that had non-MIT leader. In addition, the loss of three members, which accounted for half of the team size, at the middle of the project was another special feature of the team.

4.4.2 Goals

The first email from the project management team on November 10, 1999 via iecollab@yahoo.com to the requirement analysis team outlined the main goals of the team:

"Requirement analysts act as catalysts in identifying requirements from the information gathered from many sources, in structuring the information, and in communicating draft requirements to different audiences. Since there is a variety of participants involved in the requirement definition process, requirements must be presented in alternative, but consistent, forms that are understandable to different audiences. (Brackett, 1990)"

In addition, the importance and challenge to work with distributed team members was also stressed and the process on how to collaborate with other team members was defined as another essential learning goal of the team.

4.4.3 Work Accomplished

The requirement analysis team started to work on November 11, 1999 after it received the first email from the project management team about the project. Polo was elected to be the leader of the team. Team goals and schedule were then defined. After that, the basic tools that were used for the team were specified and the development process started.

At the beginning of the project, the project management team decided to develop the first two versions of ieCollab, meeting management and transaction management. The requirement analysis team was divided up into two sub-teams to work on the two versions simultaneously. For the ease of communication with limited time constraint, Polo, Rosa and Maria grouped together and worked for the meeting management part, while Alan, Bharath and Li-Wei worked together for the transaction management part.

The whole team continued to work on the ieCollab requirement specifications. Frequent synchronous collaborations through Microsoft NetMeeting and ICQ were held. Also,

huge number of emails was sent as the common way of asynchronous collaborations. In addition, a web repository was used to share all the documents. At first, the team used another web site (http://fciencias.ens.uabc.mx/~interact/) to share and store all the documents before the project web site was set up. After the project web site was ready, all the relevant documents were transferred to it.

The requirement elicitation process was done through the work of business and marketing managers. The requirement analysts in ieCollab gathered all the information from them and started the requirement analysis process and generated use cases. After that, the two requirement specifications were written. The requirement management process was used to monitor the work of the whole requirement analysis team.

The draft versions of requirement specification documents for both versions of ieCollab were ready for the first walkthrough on December 9, 1999. Comments from different teams were gathered for the revision on the documents during the walkthrough.

After the first walkthrough and about two months after the project started, Li-Wei, Rosa and Maria left the team at the middle of the requirement analysis process. This resulted in a significant lost of members in the team, which still had a lot of unfinished jobs to be done. After that, Polo continued to work the meeting management version while Alan and Bharath worked together on the transaction management version.

The revised versions of requirement specification documents for both versions of ieCollab were then ready for the second walkthrough on January 18, 2000. Although a lot of changes were already made after the first walkthrough, a significant number of comments of changes on the documents were received during the walkthrough.

After the second walkthrough, numerous changes and revisions were made till the end of the requirement analysis process. On February 8, 2000, the requirement analysis presentation, which aimed at presenting the concepts, process, problems and recommendations experienced during the development process, was held. Two days later, February 10, 2000, the two final versions of the requirement specification of both meeting management and transaction management were ready and it marked the end of the requirement analysis process. The whole ieCollab requirement analysis process is summarized in Figure 4-3.

At the end of the project, twenty-four items were posted on the project web site in the requirement analysis section. The most number of items posted on the web among all the ieCollab teams indicated the countless time and effort contributed by the requirement analysis team to the project.

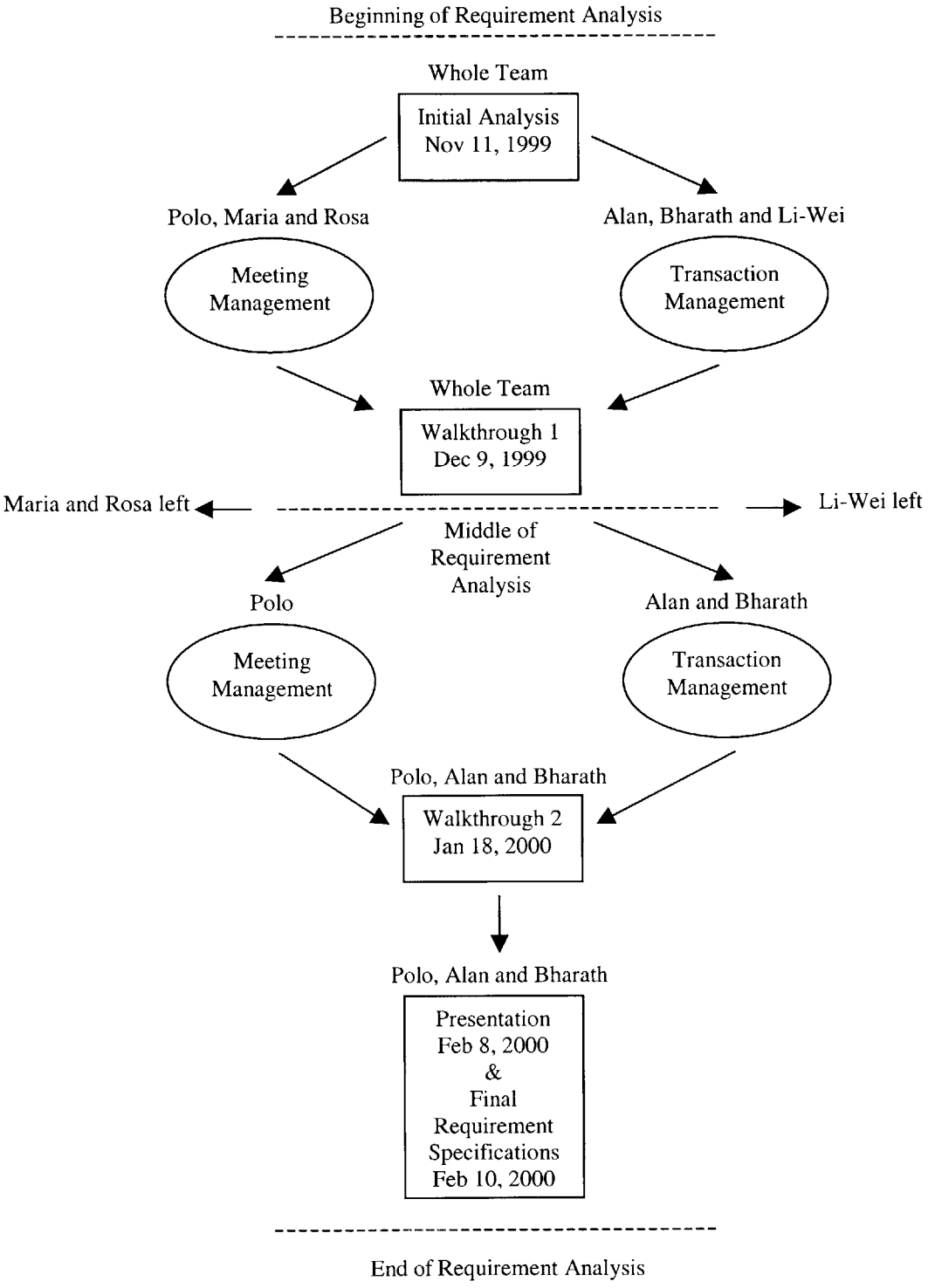


Figure 4-3: ieCollab Requirement Analysis Process

4.5 Problems and Recommendations

Although the distributed software development process has become more important and people have become more familiar with it, the seemingly easy tasks are always difficult to achieve. Various kinds of problems occurred during the development of ieCollab, they should be carefully studied and similar mistakes can be avoided through the learning of the recommendations. The common problems can be grouped to three categories: collaboration, coordination, and cooperation.

4.5.1 Collaboration

Through the ieCollab development process, collaboration was the most serious problem encountered for both co-located and distributed team members. The lack of collaboration was proven to be the root of other subsequent problems.

In the distributed teams of ieCollab, collaboration was one of the most important factors for the success of the project. As distributed team members could not meet each other physically and collaboration was further limited due to time differences among the three geographically distributed teams, the technology supporting the collaboration activities among teams became a crucial issue directly affecting the efficiency and effectiveness of collaboration.

In ieCollab, collaboration between distributed teams was always a big problem. Team members from MIT, CICESE and PUC usually had troubles trying to contact others. A lot of time was often used to set up meetings before the start of actual discussions. The connection and server problems remained the main cause of delayed meetings and inaccessibility of documents from the project web repository.

The technical difficulties experienced by distributed team members were the first problem they faced with respect to collaboration. They experienced this problem even before they could collaborate to each other. The frequent occurrence of this problem led to frustration among team members, which then lowered team spirit and morale on the ieCollab project.

As a result, the improvement on technical facilities was an urgent issue to be addressed in order to solve the first problem in collaboration among distributed teams.

Besides technical difficulties, ieCollab distributed teams also experienced many other problems but they were not as significant as the technical difficulties problem. For example, cultural and language differences often led to increased effort on collaboration to express and clarify different issues. Also, different levels of expertise and different practices on various issues seldom caused longer discussion and elaboration in the distributed environment. However, all these problems were considered as minor as compared to technical difficulties because they occurred after the prolonged delay in connection when distributed team members had already experienced enough disappointment and less time was available to carry on subsequent actual discussions and meetings.

During the development of ieCollab, it was found that collaboration problems not only occurred among distributed team members, but also between co-located members.

At the beginning of the project, there was a lack of motivation among members whose teams were not active at that time. As a result, the degree of involvement in the project varied for each member. Early in the requirement analysis phase of the project, there was a lack of collaboration between different development teams. The situation became more serious when different kinds of information or documents were requested by the

requirement analysis team from other teams. There were usually delayed responses or even no response was received at all. The lack of comments submitted to the requirement analysis team on the requirement specifications can be seen in the statistics on comments submitted by each team in Table 4-5.

Table 4-5: Number of Comments Submitted to Requirement Analysis Team

Team	Number of Comments
Project Management	0
Business Management	0.3*
Marketing Management	0
Design	3.3*
Programming	1.3*
Testing	1
Quality Assurance	2
Configuration Management	1
Knowledge Management	0

Notes: * 0.3 comment is counted when three teams submitted one combined comment

The lack of comments was due to the collaboration problem that different team members were not sure about the necessity and deadline for submitting the information. There was also uncertainty on whether they needed to post the information and comments on the web repository or not, and in which format if it was necessary. Although there was a lot of confusion on the submission of comments and other issues among teams, no collaboration took place in any way to clarify these issues. Therefore, all these problems could be easily solved if there was clear, explicit and active collaboration between teams.

4.5.2 Coordination

Other than the collaboration problem, another serious problem occurred during the development of ieCollab was coordination.

At the beginning of the project, there was a lack of software development knowledge among all the team members. Most of the team members had no prior knowledge or experience on software development. When they had to choose the roles that they played for the project, some of them did not even exactly know what the responsibilities of each role and more confusion came up when they began their work.

The coordination for ieCollab was particularly important for all the umbrella activities which complemented all the framework activities and were applied throughout the software development process. The lack of coordination among the umbrella activities led to other problems in the framework activities.

For example, the quality assurance team in ieCollab lacked a systematic way to enforce comments submission and handle comments reporting. This directly affected the project progress because comments were the valuable opinions and thoughts from all the team members. The punctual submission and gathering of these valuable resources in an easy retrieval way was particularly beneficial to the team being commented.

Regarding the documents handling in ieCollab, the knowledge management team did not have an efficient way to organize all the documents on the web repository. As a result, team members often found difficulties on finding the most updated version of different documents. Also, team members could not often access the web repository due to problems on the server. The situation sometimes persisted for a few days which directly slowed down the progress of the project, especially ieCollab was under tight schedule.

For the requirement analysis process, there was a lack of coordination between analysts and designers. There was no agreement on the requirements between analysts and designers at the beginning of the requirement analysis process. Therefore, the designers ended up starting the design process without waiting for the finish of the requirement specification. As a result, there were contradictions between analysts and designers, which led to the inconsistency between the requirement and design specifications.

All the coordination problems started from the unclear rights and responsibilities of each team, which directly affected the efficiency of each team. Redundant or unnecessary work was done because of unclear or overlap of team responsibilities. More lectures on software development process with emphasis on responsibilities of each team should be scheduled. Also, case studies on previous similar projects should be studied and discussed to avoid similar mistakes and problems occurred again.

Besides clear understanding the role of each team, project management was also a big factor in coordinating the project. Instead of waiting for the report of problems from each team, the project management team should play a more active role to coordinate all the activities between different development teams. For example, the project management team should coordinate meetings between analysts and designers to solve any conflicts between them. Moreover, it should ask for the plans from each team at the beginning of the project to clearly define the work and schedule. Furthermore, more enforcement power should be given to the project management team to strictly observe the deadline on the project.

4.5.3 Cooperation

Besides collaboration and coordination problems, cooperation was another key problem in ieCollab. Without cooperation between team members, collaboration and coordination processes could not be carried out smoothly.

The lack of interaction between different teams was the main obstacle on cooperation. Team members did not know each other very well at the beginning of the project, even though there was a web page creation exercise to promote friendship among them because that was limited between the group of two people who created the web pages for each other. Although everyone was excited about the project at the beginning, it lacked a team spirit and cohesion when team members did not know each other very well. This lack of interaction also happened in co-located teams, and became more noticeable in distributed teams. More casual contact or meetings should be held outsides classes to help to know each member better.

Besides the lack of interaction, the cooperation problems could also be reflected by the submission of documents and comments. After the correct procedures of handling documents and comments were clearly explained, there still remained the problem of delayed submission. The lack of cooperation on comments submission for requirement specifications before walkthroughs contributed significantly to the frequent changes of the specifications later during the requirement analysis process. Also, due to the delayed comments, new comments were often received at different times after certain updated versions of the documents were already posted on the web repository. This was another factor that led to the unexpectedly large number of different versions of the requirement specifications.

To address these problems, the project management team should be notified immediately when certain problems arose and subsequent actions could be taken to prevent the occurrence of similar incidents in the future. Also, frequent feedback should be asked from other teams and a system for evaluating the quality of comments should be setup to ensure the awareness and involvement of other teams on the work of requirement analysis team.

4.5.4 Others

There were many other problems faced by the ieCollab development teams besides collaboration, coordination, and cooperation problems. All of these problems had to deal with the background knowledge of team members or the setting of the project.

Most of the team members had civil engineering undergraduate degrees with little computer science or language background. Although some team members had computer science background or more advanced knowledge on different computer languages, there was still a lack of technical knowledge in general. For example, the programming team classified several computer skills needed for ieCollab during its presentation. There was good level of knowledge on Java, fair level on Java Database Connectivity (JDBC) and Structured Query Language (SQL), and limited level on Common Object Request Broker Architecture (CORBA) and Oracle. This limited technical background of team members could be improved through lectures before the start of project or small courses offered in the three schools in January when no formal project meetings and lectures were held.

About the project structure, there was an unbalanced number of team members among the three distributed teams. More than two-thirds of the ieCollab team members were from MIT at the beginning of the project, and over ninety percent of the whole team comprised

of MIT students at the middle of the project. This uneven distribution of distributed team members created an atmosphere of internal discussion within MIT teams without the involvement of members from CICESE and PUC. Also, all development teams besides the requirement analysis team had MIT members as team leaders. This created the center of authority in MIT and decisions were often made in MIT without discussing them with members from the other two locations. Even in the requirement analysis team, although the team leader was from CICESE, most decisions were made within the MIT team members. The situation became more serious after the middle of the requirement analysis process, when two of the PUC members left the project and continuous connection problems occurred for meetings.

Besides the team structure of ieCollab, the major loss of team members under limited project time was another problem because of the settings of the project. Over thirty percent of the ieCollab team left the project at the middle of the development stage, in which all the PUC members left the project. Under the nine-month development time schedule, this was a huge impact both in terms of personnel and team spirit. The significant decrease in the number of team members created excessive pressure on continuing team members. Many of them believed the project could not be finished when there was the sudden drop in the number of team members. This directly affected the team spirit in a negative way.

These problems could be solved by creating an equal number of team members in each distributed team and team leaders should be chosen from members in different locations. In addition, all team members should be encouraged to stay from the beginning till the end of the project to improve the effectiveness of the development process of the project.

4.6 Summary

Despite all the problems occurred during the development of ieCollab, all the team members agreed that the distributed learning experience through collaboration with each other was one of the most valuable experiences gained from this project. Not only the software development process and the roles in each development team were studied and practiced through ieCollab, but also the importance of teamwork and collaboration in distributed software development projects was recognized. At the end of the project, all the team members considered the seemingly painful experience from all the problems as an unforgettable one in their lives.

Chapter Five

Conclusion

This thesis concludes with improvement and suggestions for requirement analysis. As it has been shown, this phase of the software development process should not be taken lightly, and therefore its potential enhancements are important as well. Therefore, this last chapter focuses on factors for successful development and the future work on distributed requirement analysis process.

5.1 Success Factors

In order to have a successful requirement analysis for distributed software development, four areas need to be considered: people, product, process, and technology. The importance of all four of these factors should not be overlooked and the lack of concentration on either one of them can lead to failure of the distributed requirement analysis process.

5.1.1 People

Out of the four important areas, the most critical factor for the success of distributed software development is the people factor (Brooks, 1987).

Software development is the practice of software engineers who transform their knowledge into software products. Improving the software development process requires continual improvement of its developers and of the conditions that improve their performance in the knowledge-intense software industry. Research shows that there is greater than 10 to 1 differences in productivity among individuals with different depths and breadths of experience and there is 5 to 1 differences in productivity among groups with different levels of experience (McConnell, 1996).

The social and technical aspects of the requirement analysis process need well-rounded people with different skills to cope with the complicated nature of this very first step in software development process. The success of the requirement analysts not only improves the cost and time of the software project, but also increases the morale and spirit of the whole development team.

In the distributed software development of the requirement analysis process, the management of intellectual complexity on the large-scale integrated work is essential. The change from creative and talented individuals to motivated and productive teams that emphasizes continuous learning is a necessity to improve the development results.

As illustrated in previous chapters as well as in real life examples, all kinds of problems have direct or indirect associations with people. Collaboration, coordination, and cooperation problems all have to do with people. A lot of these problems can be avoided

if better people with suitable skills, teamwork spirit and strong motivation are involved in the development process.

5.1.2 Product

At the start of the software project, clear, well-defined objectives and scope should be unanimously agreed upon by both the software developers and customers. After that, alternative solutions are considered based on cost estimates, time schedule, and the team structure for different project tasks.

The requirement analysis process takes place during the early phase of the software development process, when most solid information is not available. Therefore, a detailed software requirement specification would provide a lot of useful information for understanding the product.

The distributed development environment of the requirement analysis process would definitely increase the difficulties on product definition. On the other hand, distributed team members would have different background and culture, knowledge and expertise that help to define the product from a different view. The resulting requirements are generally more diverse and complete to meet the clients' needs.

From the case study, it can be seen that the project scope is very important for distributed requirement analysis and software development. Unclear or unrealistic product definitions based on incorrect cost, time and labor estimations can ultimately lead to lengthened requirement analysis process or even project failure. Therefore, careful product definition at the beginning of the software development project should not be overlooked.

5.1.3 Process

Many different software development processes are used for various kinds of software development projects. The correct selection of the software process that is appropriate for the specific team and target software is the main concern to reduce development cost, time, and defects.

Being the first phase of the software development process, the requirement analysis process often has more problems on the process issues. Requirement analysts often encounter the inappropriate selection of the software development process before another proper process is chosen. The switching from one process to another sometimes leads to an increase of development effort for the requirement analysis process.

The distributed development of requirement analysis would complicate the process selection in various aspects. The selection of one process model over another usually involves taking various kinds and degrees of risk. There is no particular process for an organization to build a software. The selection can be based on different factors such as the nature of the software to be built and the resources available. Furthermore, the geographically distributed nature of team members can often lead to repetitive or redundant work if an inefficient model is selected. Therefore, proper management should be used to monitor the process in order to increase the development speed.

The change in process selection from the case study is a good example to show how it affects the requirement analysis process. The change, which takes place during the requirement analysis process, affects the coordination between analysts and designers and the change in schedule for both teams. Proper process selection by the project management team should be done to ensure the smooth and effective development process.

5.1.4 Technology

The fast emergence of new technology certainly improves the software development process in different ways. The new development tools being used by different developers can lead to an increase in development speed and a decrease in development effort.

New technology helps the requirement analysis process in various ways. The new communication technology can be used for the elicitation process between analysts and clients to compromise the lack of time and involvement of clients. Also, the extensive use of use cases, which keeps improving in different versions, can help to clarify requirements in diagrams and textual representations.

The use of technology can also improve the collaboration process in distributed requirement analysis process in many different ways. It helps to shorten setup time for collaboration between distributed team members and reduce the effort required for the connection and various technical problems. The improvement in technology also makes real-time collaboration no longer a hassle for distributed software development.

However, the cost of implementing new technology tools is always a problem. The advanced technology may not be available in all distributed areas. The cost of getting the equipment may far exceed the budget of the software project. As seen from the case study, technology becomes a big problem for collaboration instead of contribution because of unavailability or unpopularity.

5.2 Future Work

The future of requirement analysis for distributed software development should not just focus on the currently essential four factors in the previous section. Over the next decade, two major areas: reuse and retooling, will likely be the dominating issues.

5.2.1 Reuse

Reusability has been suggested to be the key to improve software development productivity and quality. There are other suggestions that reuse at the requirement analysis process can increase reuse at later stages of development. Although the importance of requirement reuse is noticed by the software industry, little evidence shows that requirement reuse is widely practiced (Lam et al, 1997).

However, in recent software development process, techniques for software reuse became the mainstream instead of developing the software from scratch, especially after the rapid adoption of the object-oriented development approach. There are certain areas in the requirement analysis process that could be easily reused, as listed below (Saeki, 1999):

Interview patterns and questionnaire patterns:

During the requirement elicitation process, requirement analysts have interviews and questionnaires to elicit the requirements with the users and clients. Similar patterns for interviews and questionnaires can be easily reused if the patterns are constructed carefully.

Template for describing use cases:

Previous defined use cases can be reused to assist requirement analysts to describe use cases. The structure of the basic template has slots that requirement analysts have to fill with new answers which are collected from the elicitation process.

Use case pattern:

During the modeling process, requirement analysts find structured use cases with patterns that can easily be reused. Structured use cases consist of those with classes and generalization relationships that provide reusable and changeable structure.

Reuse for requirement process is not limited to the above three examples. An increase in requirement reuse can be achieved using relatively inexpensive and simple ways that do not require radical organizational changes (Lam et al, 1997). With the increasing challenge in distributed development environment, different areas of reuse should be explored to increase development speed and decrease cost and labor. As a result, software reuse should be one of the major future focuses on distributed requirement analysis process.

5.2.2 Retooling

The software development process is a human-intensive process. Although improvements on the development process were made gradually through human capabilities in the past, the use of more powerful tools must be increased to make significant improvement (Humphrey, 1989).

Computer-aided software engineering (CASE) systems have been used by software engineers to meet this demand for a long time. CASE tools are used to automate the software process in order to improve the quality and productivity of the software development process (Humphrey, 1989).

However, current CASE tools are not adequate for the requirement analysis process. Most of them are generally software engineering oriented and there is a lack of particular CASE tools for requirement analysis process. The tools that are developed for requirement analysis usually have many problems and therefore are not widely used. Also, there is a lack of tools that supports group work or helps facilitating the process. The supply of suitable technologies for distributed work cannot satisfy the increasing demand (Bubenko, 1995).

In recent years, there has been an increasing concern about the need of CASE tools for requirement analysis. For example, a CASE tool for Specification, the Trade-off, and the Analysis for the Requirements (STAR) was introduced to assist requirement analysts in identifying conflicting requirements and in accessing clients' trade-off preferences among conflicting requirements (Yen et al, 1998). Another example is the Goal Based Requirements Analysis Tool (GBRAT) which uses interactive world-wide-web technologies to support the collaborative nature of the requirement analysis process. It provides procedural support for identifying, elaborating, and organizing goals for the requirement specification that can be viewed and modified by the distributed team members around the world (Anton et al, 1996).

However, to develop a suitable and useful requirement analysis tool, developers need to study carefully the special needs of requirement analysts. While better tools cannot ever replace requirement analysts, poor ones can often impede and frustrate them. Therefore, careful selection of requirement analysis tools is a key to successful retooling process.

Through the recent study and success of retooling, its development should be another future focus for the improvement of the distributed requirement analysis process.

5.3 Final Remarks

It is hoped that this thesis provides readers with a greater understanding of the requirement analysis process in the distributed software development environment. Also, it is important to know and realize its considerable significance on software projects as well as gain more knowledge of how to perform the analysis for continued success. This knowledge needs to be augmented with good collaboration, coordination, cooperation and an eye toward the future of reuse and retooling.

Bibliography

[1] Anton, A. I., Liang, E. and Rodenstein, R. A., "A Web-based Requirements Analysis Tool." *Proceedings of the 5th Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises 1996*, 1996, pp.238-243.

[2] Aoyama, M., "Distributed Concurrent Development of Software Systems: an Object-Oriented Process Model." *Proceedings of Fourteenth Annual International Computer Software and Applications Conference*, 1990, pp. 330-337.

[3] Blum, B. I., *Software Engineering: A Holistic View*. Oxford University Press, New York, 1992.

[4] Brackett, J. W., *Software Requirements*. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1990.

[5] Brooks, F., "No Silver Bullet: Essence and Accidents of Software Engineering." *Computer*, April 1987, pp. 10-19.

[6] Bruegge, B. and Dutoit, A. H., *Object-Oriented Software Engineering: Conquering Complex and Changing System*. Prentice Hall, Upper Saddle River, NJ, 2000.

[7] Bubenko, J. A. Jr., "Challenges in Requirements Engineering." *Proceedings of the Second IEEE International Symposium on Requirements Engineering*, 1995, pp. 160-162.

[8] Davis, J. S., "Identification of Errors in Software Requirements through Use of Automated Requirements Tools." *Information and Software Technology*, Vol. 31, No. 9, November 1989, pp. 472-476.

[9] Dorfman, M. and Thayer, R. H., *Software Engineering*. IEEE Computer Society Press, Los Alamitos, CA, 1997.

[10] Dorfman, M. and Thayer, R. H., *Standards, Guidelines, and Examples on System and Software Requirements Engineering*. IEEE Computer Society Press, Washington, DC, 1990.

[11] Firesmith, D. G., *Object-Oriented Requirements Analysis and Logical Design: A Software Engineering Approach*. Wiley, New York, 1993.

[12] Fowler, M. and Scott, M., *UML Distilled*. Addison-Wesley, Reading, MA, 1999.

[13] Haag, Z., Foley, R. and Newman, J., "Software Process Improvement in Geographically Distributed Software Engineering: an Initial Evaluation." *Proceedings of the 23rd EUROMICRO Conference*, 1997, pp. 134-141.

[14] Herlea, D. and Greenberg, S., "Using a Groupware Space for Distributed Requirements Engineering." *Proceedings of Seventh IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 1998 (WET ICE '98)*, 1998, pp. 57-62.

[15] Humphrey, W., *CASE Planning and the Software Process.* Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1989.

[16] IEEE Computer Society, "IEEE Recommended Practice for Software Requirements Specifications." *IEEE Std 830-1998.* IEEE, New York, 1998.

[17] IEEE Computer Society, IEEE Standard Glossary of Software Engineering Terminology. IEEE, New York, 1983.

[18] Jalote, P., *An Integrated Approach to Software Engineering.* Springer, New York, 1997.

[19] Jirotka, M. and Goguen, J., *Requirements Engineering: Social and Technical Issues.* Academic Press, San Diego, CA, 1994.

[20] Johansson, C., Dittrich, Y. and Juustila, A., "Software Engineering across Boundaries: Student Project in Distributed Collaboration." *IEEE Transactions on Professional Communication*, 1999, pp. 286-296.

[21] Lam, W., McDermid, J. A. and Vickers, A. J., "Ten Steps Towards Systematic Requirements Reuse." *Proceedings of the Third IEEE International Symposium on Requirements Engineering*, 1997, pp. 6-15.

[22] McConnell, S., *Rapid Development.* Microsoft Press, Redmond, Washington, 1996.

[23] Medina Sanchez, A., *Software Requirements Analysis for a Data Acquisition System: A Structured Approach.* Thesis (M.S.) -- Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 1993.

[24] Mills, J., *ieCollab Business Plan version 1*, Department of Civil and Environmental Engineering, Massachusetts Institute of Technology, Cambridge, MA, 2000.

[25] Patterson, B., *The Requirements Definition Phase in the Development of Software Applications*. Thesis (S.M.M.O.T.) -- Massachusetts Institute of Technology, Sloan School of Management, 1999.

[26] Paulk, M., Curtis, B., Chrissis, M. and Weber, C., *Capability Maturity Model for Software, Version 1.1*. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1993.

[27] Paulk, M., Weber, C., Garcia, S., Chrissis, M. and Bush, M., *Key Practices of the Capability Maturity Model Version 1.1*. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1993.

[28] Pressman, R. S., *Software Engineering: A Practitioner's Approach*. McGraw-Hill, New York, NY, 1997.

[29] Saeki, M., "Reusing Use Case Descriptions for Requirements Specification: Towards Use Case Patterns." *Proceedings of the Sixth Asia Pacific Software Engineering Conference*, 1999, pp. 309-316.

[30] Salo, A. and Kakola, T., "Groupware Architecture for Requirements Processes in New Product Development." *Proceedings of the 32nd Annual Hawaii International Conference on Systems Sciences 1999*, 1999, pp. 1-9.

[31] Steinfield, C., Jang, C. and Pfaff, B., "Supporting Virtual Team Collaboration: The TeamSCOPE System." *Proceeding of the International ACM SIGGROUP Conference on Supporting Group Work*, 1999, pp. 81-90.

[32] Thayer, R. H. and Dorfman, M., *Software Requirement Engineering*. IEEE Computer Society Press, Los Alamitos, CA, 1997.

[33] Weinberg, V., *Structured Analysis*. Prentice-Hall, Englewood Cliffs, NJ, 1980.

[34] Yen, J., Yin, J., and Tiao, W. A., "STAR: A CASE Tool for Requirements Engineering." *Proceedings of 1998 IEEE Workshop on Application-Specific Software Engineering Technology*, 1998, pp. 28-33.

Appendices

Appendix A

Appendix B

Requirements Specification For ieCollab Version 1 Meeting Management

Specification Version 1.4

Requirement Analysis Team

Update from Version 1.3 by Bharath Krishnan, Alan Ng
Date: February 17, 2000
Participants on Modification:
- online Sessions: Bharath, Polo
- offline Sessions: Bharath, Polo, Alan, Wassim (Design Team)

References and Links
(All references are stored at http://collaborate.mit.edu/1.120.html)

• Prioritized Requirements v1.0 for ieCollab	November 25, 1999
• Initial Requirements Specification v0.4 for ieCollab Version 1	November 26, 1999
• Requirements Specification v1.1 for ieCollab Version 1	January 19, 2000
• Requirements Specification v1.2 for ieCollab Version 1	January 22, 2000
• Requirements Specification v1.3 for ieCollab Version 1	February 9, 2000

Outline

1. Introduction

This document presents the Use Case Model for ieCollab Version 1. This version mainly focuses on the Meeting Management Environment, which allows distributed users to setup and manage online Meetings. ieCollab will be developed in four versions/phases:

Meeting Management Environment (v.1),
Transaction Management (v.2),
Collaboration Server (v.3), and
Application Server (v.4).

A separate document presents the functional requirements specification for ieCollab Version 2.

The purpose of this document is to present in a formal language the required functionality of ieCollab for the Meeting Management Environment, establish what the system does and not how it will be implemented. This includes functionality to keep track of Meeting agenda, Meeting participants, user profiles, and Meeting styles (templates). In addition, it will serve as a reference guide for the design and development of ieCollab.

2. Architecture, Use Case Diagrams and Description

2.1 Overview

The basic functionality of ieCollab will be implemented in this version, it will be able to allow distributed users to setup and manage online Meetings through a Meeting Management Environment which includes the necessary mechanisms to keep track of Meeting agenda, Meeting participants, user profiles, and Meeting styles (templates).

2.2 General Architecture

The major components of the architecture in this version are depicted in Figure 1.

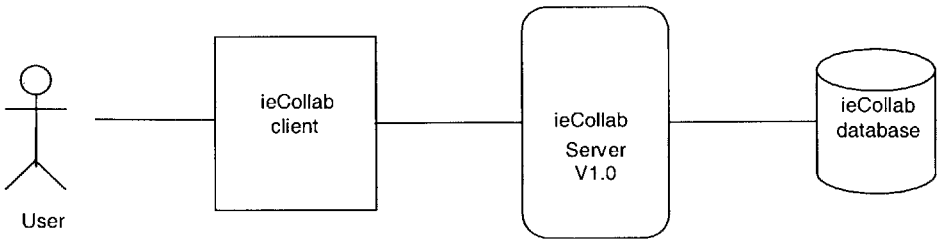


Figure 1. ieCollab basic architecture.

The ieCollab client will be in charge of obtaining requests from and presenting the results to the user. It will require no local (or minimal) processing to do this job.

The ieCollab Server Version1.0 will be responsible for answering the client requests and for all information processing in the system.

The ieCollab database will be in charge of storing all the important information (user, Workgroup and Meeting information).

2.3 Actors

Actors are used to identify the functionality of the system. Before giving a description of the identified actors, the following terms must be defined:

User account: It is a record to hold the system specific information related to a user. The login name and the password are examples.

User profile: It is a record to hold the non-specific system information related to a user. The user profile holds most of the information gathered from the user during registration.

Some of the attributes for these elements are defined in Appendix B. Once these terms are defined, a description of the different actors identified for this version follows.

External user
This user has not been registered into the system yet. He has neither a user profile, nor an account.

Normal user
This is a registered user. He has an account and a user profile.

Workgroup leader
This is a role that a normal user can acquire. It is achieved by creating a Workgroup, or after being granted it by another leader of that Workgroup. This role will persist across Meetings.

Normal Workgroup member
This is another role that a normal user can acquire. It is achieved after the user has been added to a Workgroup. This role will persist across Meetings.

Meeting leader
This is a role that a normal user can acquire. It is achieved by scheduling a Meeting, or after being granted by the previous Meeting leader. At all times, there should be only one Meeting leader for a Meeting. This role will not persist across Meetings. Every time a new Meeting is called, this role should be assigned to a normal user.

Meeting participant
This is a role that a normal user can acquire. It is achieved after being invited/accepted into a Meeting. This role will not persist across Meetings. Every time a new Meeting is called, this role should be assigned to a normal user.

The System
This is the server. This actor is used to identify tasks that the server should autonomously execute.

2.4 Use Cases for the ieCollab System

Use cases represent typical interaction between a user and the specified system. The level of abstraction of them may vary, among other things, depending on how early or late in the development process they are presented.

2.4.1 Registration

The External users start this use case. It provides the capability to register a new user, create a new user profile and a new user account.

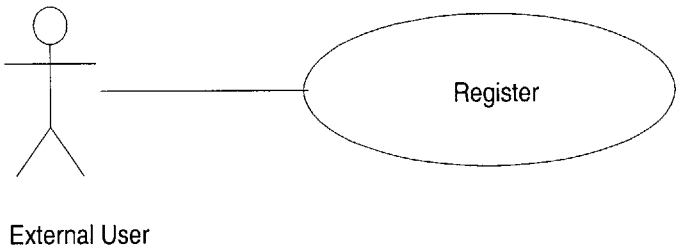


Figure 2: The Registration Use Case

2.4.2 Login

The Normal users start this use case. It provides the capability to enter into the system and use its resources.

2.4.3 Logout

This use case represents the direct opposite of the previous use case.

2.4.4 Search

This is the functionality to search the ieCollab database for users or Workgroups.

2.4.5 Edit User Profile

This use case lets the user change his profile information.

See Figure 3.

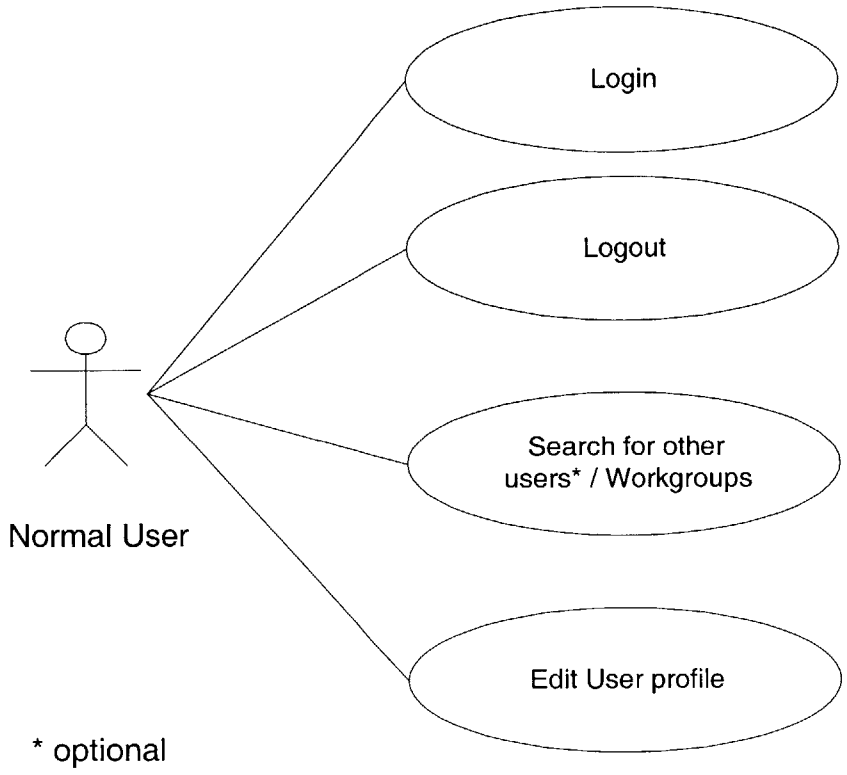


Figure 3: Normal User Use Cases

2.4.6 Workgroup management

The use cases here can be divided into those used by a Normal User, Normal Workgroup Member and those used by the Workgroup Leader.

For the Normal User, the use cases which relate to Workgroup management are (Figure 4)

- List Accessible Workgroups.
- Search for Workgroups by name and by description.
- Request Workgroup membership.
- Accept or Decline a request to join a Workgroup.
- Create a new Workgroup.

For the Normal Workgroup Member: (Figure 5)

- Relinquish Workgroup membership.
- List members of Workgroup.
- List scheduled Meetings for Workgroup.

- List and access previous Meeting logs for Workgroup.

For the Workgroup Leader: (Figure 6)
(The leader can invoke all the use cases pertaining to Normal Workgroup Member except "relinquish membership")

- Add new member to Workgroup
- Accede to or Deny request for membership
- Invite a user to join a Workgroup
- Revoke Workgroup Membership
- List / Remove old Meeting logs
- Change Workgroup leader. (Only then can the leader relinquish his membership).
- Remove Workgroup
- Request Workgroup membership for her Workgroup.

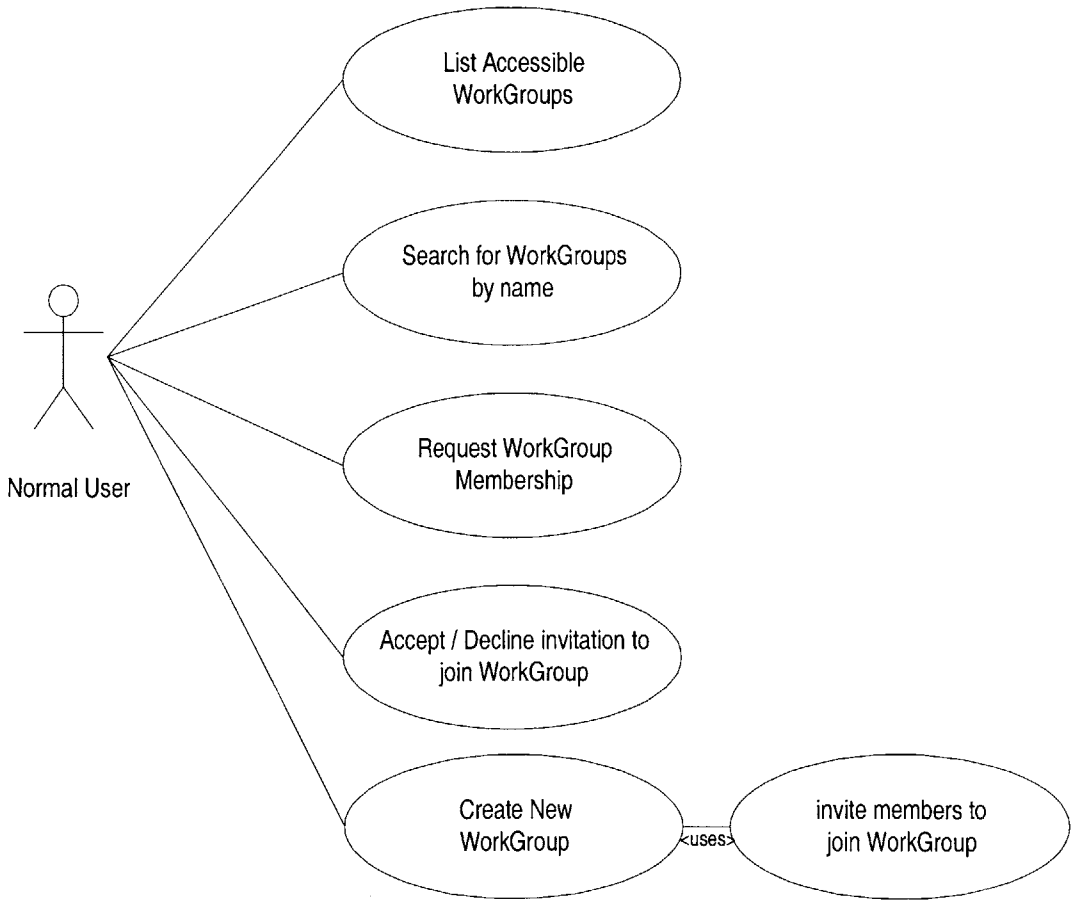


Figure 4: Use Cases related to Workgroups for Normal User

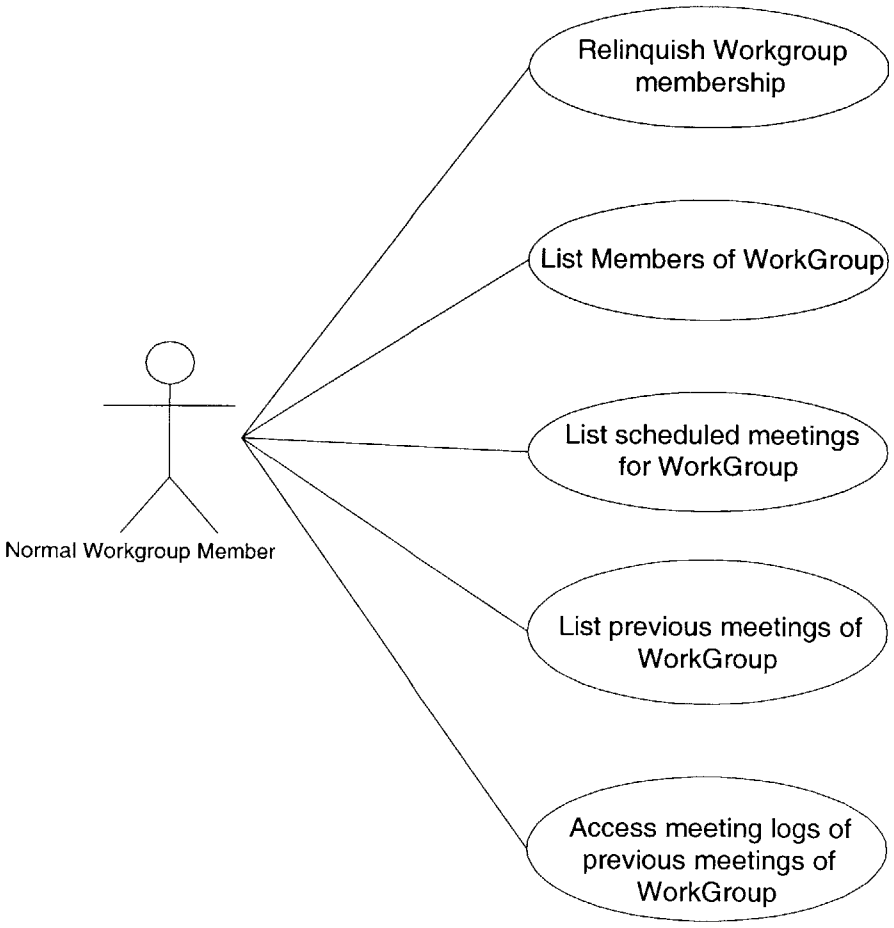


Figure 5: Use Cases for Normal Workgroup Member

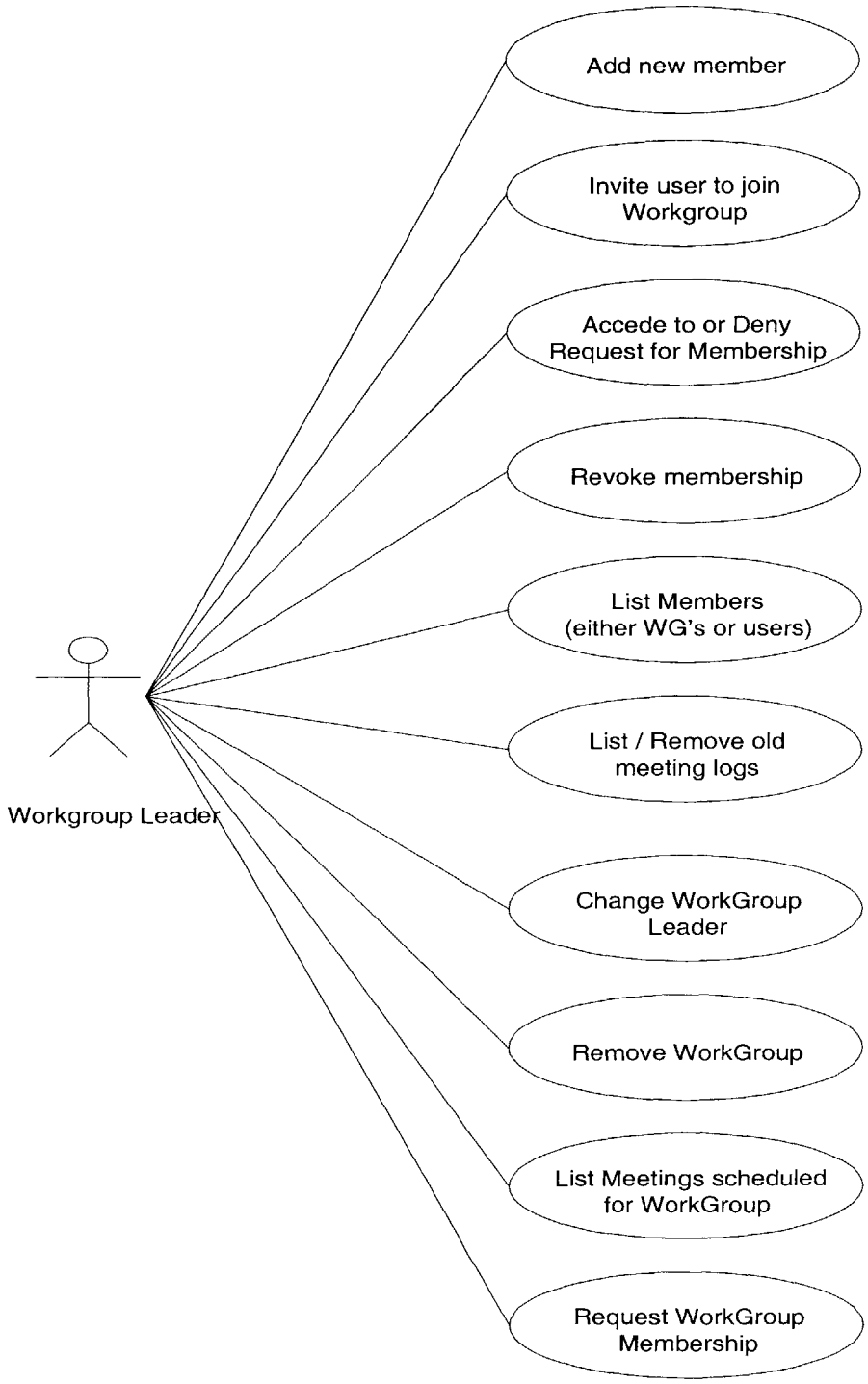


Figure 6: Use Cases for Workgroup Leader

2.4.7 Meeting management (Figures 7 and 8)

The Normal User can schedule a Meeting and become the leader for that Meeting. The relevant use cases are:

- Schedule a new Meeting
- Set meeting Agenda.
- Set meeting roles.
- Select a Meeting template.
- Accept or Decline invitation to join a meeting
- Start a scheduled meeting
- Request to join a meeting

The Meeting leader has the following use cases.

- Cancel a scheduled Meeting. (A meeting cancellation message is sent to the user. This will appear in the Invitation List and can be clicked away. Also the graphical representation (color or icon associated with that specific meeting changes)
- Change the agenda, participants and scheduled time for a Meeting. (Edit meeting information)
- Option to save Meeting logs.
- In addition the leader has all the use cases associated with a Normal Meeting Member. (start a meeting is an example)

Also any user invited to a Meeting can

- Start the Meeting. The Meeting can be started within a small interval of time before it is scheduled to begin (say 10 minutes)

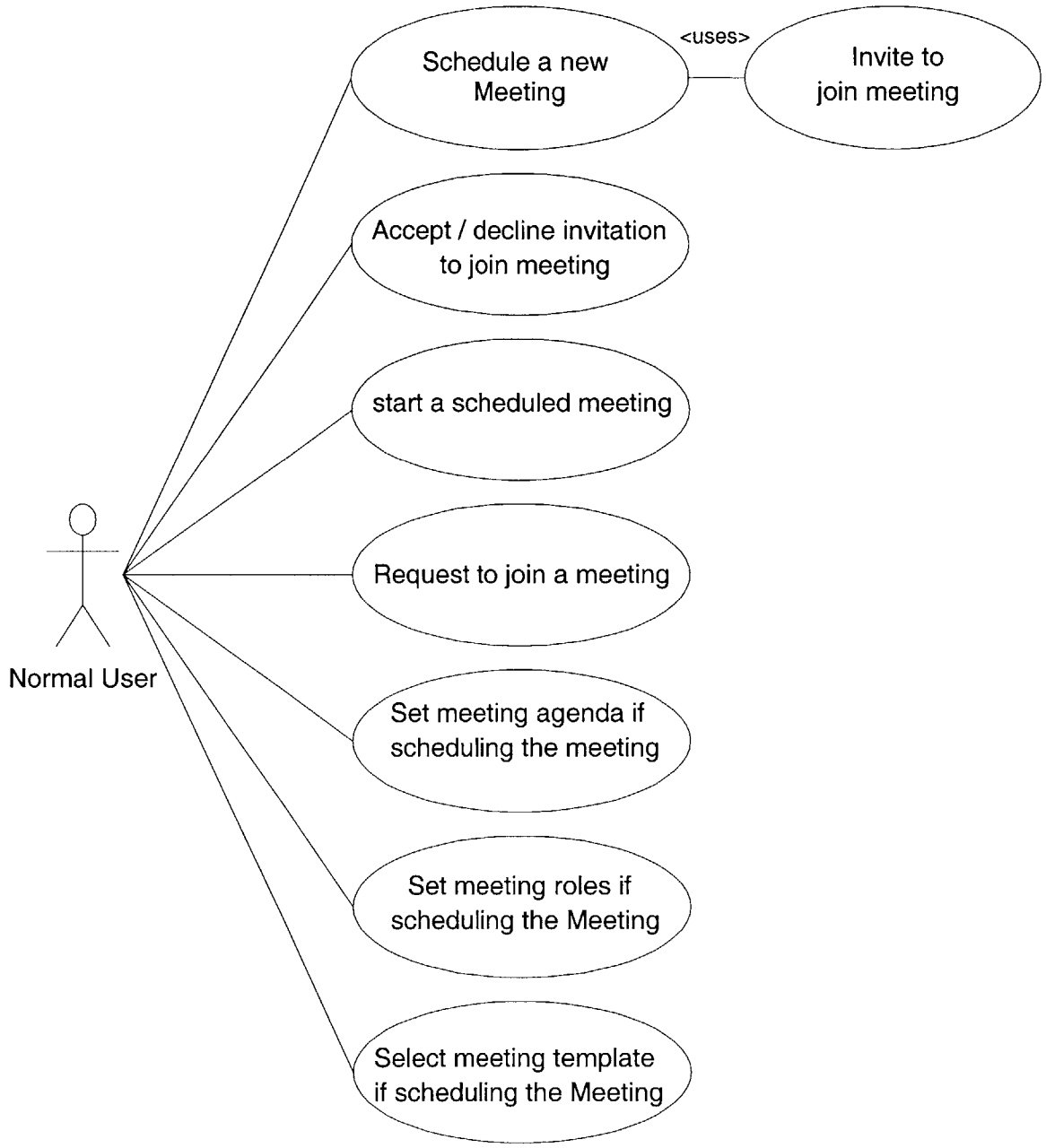


Figure 7: Meeting related use cases for Normal User

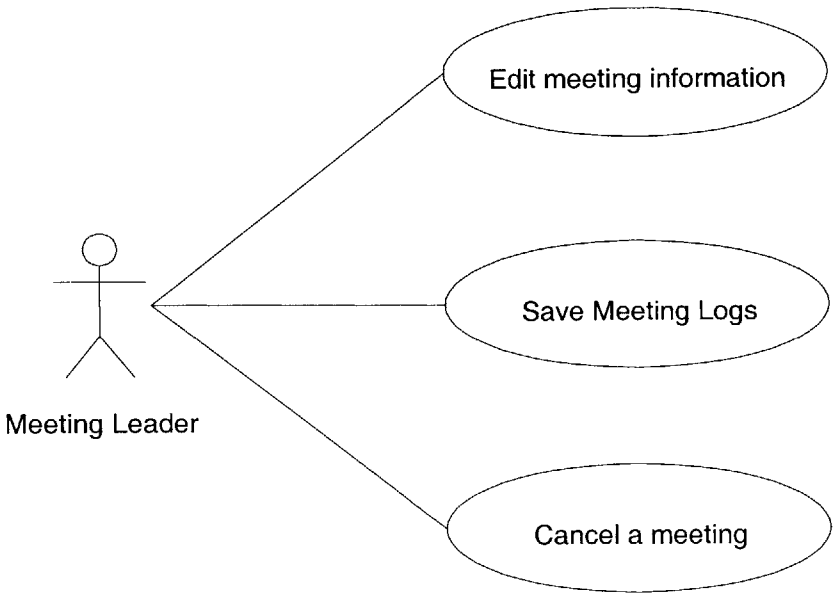


Figure 8: Use Cases for Meeting Leader

2.4.8 System maintenance

This includes use cases for the server to do periodically.

- Removing Meeting logs from the database after checking that all links to them have been deleted.

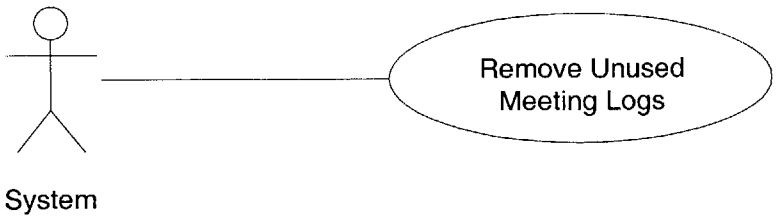


Figure 9: Use Case for system maintenance

2.5 The Workgroup

A Workgroup is an aggregation mechanism to keep users who need to work together under a common name and identity. This grouping mechanism provides the ability to have operations that work on several users at the same time. Operations such us Meeting scheduling, and permission granting or denial on certain resources, are examples of it. The main difference between a Workgroup and a group of people invited to a Meeting is that the Workgroup will persist across Meetings, whereas the group invited to a Meeting will cease existing as soon as the Meeting is over. Workgroups may contain other Workgroups in addition to normal users.

2.6 Meeting Management

Meetings are an essential part of this environment. The following terms are relevant to Meeting management.

2.6.1 Meeting: A Meeting is a gathering of users to work on a specific topic or activity, following a specific set of steps and using a specific set of tools.

2.6.2 Meeting agenda: It is a formulated plan listing a set of topics to be discussed or a set of activities to be done (usually) in a chronologically order. It is created specifically to satisfy or fulfill certain requirements. The agenda usually specifies the topic for discussion and the leader for that topic in chronological order.

2.6.3 Meeting template: The Meeting template serves to decide the structure of the Meeting. Examples would be a *free for all* Meeting in which everyone is allowed to speak and the leader does not have any veto power or a *chaired* Meeting where participants have to get the chairman's permission before they can speak.

Some of the attributes for these elements are defined in Appendix B.

3. GUI Description

Appendix A shows the prototype Graphical User Interface for the ieCollab client.

The idea behind the GUI is to keep things simple. Instead of offering a plethora of options to the user, we have a minimalist interface with a lot of options hidden behind buttons and mouse clicks.
Take for example the list of Workgroups. A mouse click (right click) pops up a menu which gives the user the Workgroup management options.

3.1 Description of GUI Components

(Refer to Figures 10, 11 and 12)

3.1.1 Buttons

3.1.1.1 The edit user profile button

pops up a menu with text boxes where the user can change his profile. The user profile details are indicated in Appendix B.

3.1.1.2 The schedule Meeting button

pops up a menu which allows the user to schedule a new Meeting. The options in this window include the following:

- Set the Meeting date, time and an identifier
- Set the Meeting leader (default is the person who initiates the process itself).
- Set Meeting participants. (Both users and Workgroups can be included) The same search tool which is present in the main window (after login Figure 11) can be present here so that the user can search for the Workgroups or users he has to include in the Meeting.
- Choose Meeting template selection from list of provided templates.
- Set Meeting agenda.

3.1.1.3 The create Workgroup button

pops up a menu which allows the user to create a new Workgroup. The options in this window include the following:

- Set the Workgroup name
- Set the Workgroup description (A string which describes the Workgroup)
- Set the Workgroup's members which can be either users or other Workgroups. Here too the search tool can be present. Once the Workgroup is created, an invitation is sent to the members.
- Set the Workgroup leader, which by default is the person who creates the Workgroup. This person is automatically made a member of the Workgroup, i.e. he does not get an invitation to join the group.
- Set the Workgroup's privacy policy. There can be three levels here.
  - Public: Visible in searches and anyone can join
  - Protected: Visible in searches but you need permission before you can join

- Private: Invisible in searches and permission required before joining in

3.1.1.4 The Logoff button

Just logs the user out of the system. There is a peculiar scenario here which should be handled. If the network connection between the client and the server is disrupted, the user should still be able to close the client window nicely by using the logout button. When the connection comes back online and the server finds that the client does not exist anymore, it should perform a clean up of the resources allocated to that particular client.

3.1.1.5 The Add Workgroup button

This button allows the user to add a Workgroup to his list if he knows the Workgroup name. If the name is incorrect, a search is performed and the user gets a list of matches.

3.1.1.6 The Search Tool

The search pops up a menu with a limited (10) number of closest matches to the user's search criterion. The criteria can be a username, Workgroup description or Workgroup name. Additional criteria can be added if needed. The criterion is selected from the drop down list next to the text box where the search string is entered.

Operations can be performed on the search results. For example, if Workgroups are the search result, the user gets the option of sending a request to join that Workgroup (or if it is a public Workgroup, he can join straight away).

3.1.2 The Lists

3.1.2.1 The Workgroup List

This is the list of Workgroups that the user is a member of. Different kinds of Workgroups are differentiated by their appearance in the list. This difference can take the form of different colors or icons for Workgroups according to whether the Workgroup is Private, Protected or Public. The items in this list are dynamic, i.e., a mouse click on one of the items will pop up a menu of Workgroup management operations. For a normal Workgroup member these include the use cases described in Figure 5. For the Workgroup leader, this includes all the use cases described in Figure 6.

3.1.2.2 The Meeting List

This is the list of scheduled Meetings for the user. The list will include all Meetings scheduled for the user within a specific period of time in the future (10 days). The Meetings can be either those to which the user has been invited personally or those to which he has been invited through a Workgroup. The items in this list too are dynamic in the sense that clicking on them pops up menus. These menus the use cases described in 2.4.7.

3.1.2.3 The Old Meeting Log List

This is a list of old Meeting logs which the user was invited to personally. The Meeting logs for Meetings which the user attended as part of a Workgroup will be presented in the old Meeting log part of the Workgroup list pop up menu. The Meeting logs will include information like

- Date, time and topic for the Meeting
- Participant list
- Logs of any discussions held

3.1.2.4 The Invitations List

This is a list of the invitations the user has received asking her to join a Workgroup or Meeting. The user can either accept or turn down an invitation by using a menu which pops up when an invitation is clicked. The invitation should include information about the Meeting, its participants, agenda etc.

An additional entity which can pop up here is a Meeting cancellation message.

4. Conclusion

In the model provided, the focus is on the functional specification. The use cases are not scenarios, because they are not intended to be specific records of a detailed set of interactions, instead, they are general specifications that capture the critical steps of the user-system interactions.

The model refers to the basic functionality of ieCollab version 1, which is to provide a Meeting environment. Later versions shall increment the initial functionality and will focus in the implementation of a transaction scheme based on ASP model (v.2), a collaboration space (v.3) and finally a full-fledged application server (v.4).

Appendix A. Prototype GUI

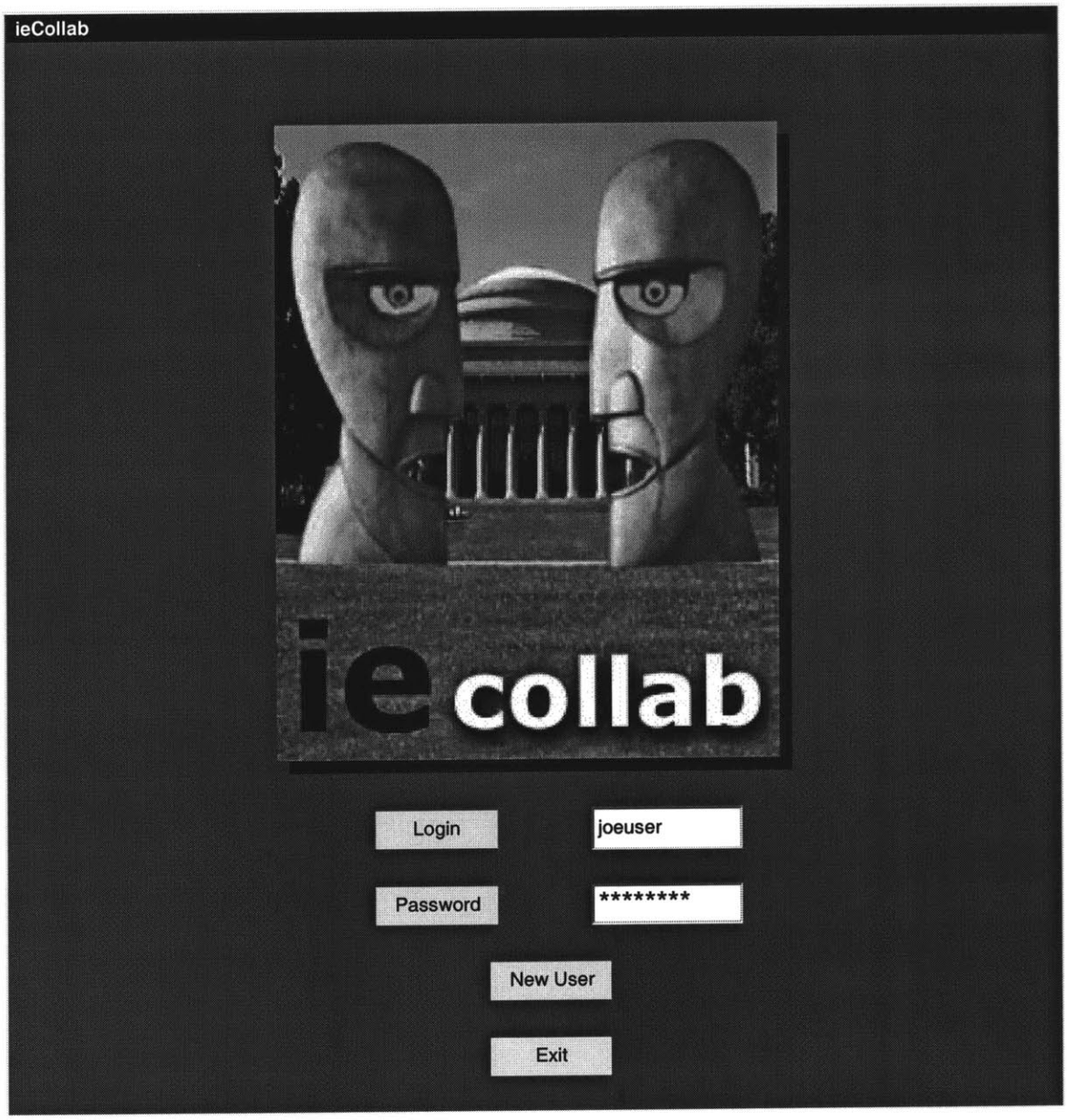


Figure 10: Startup screen

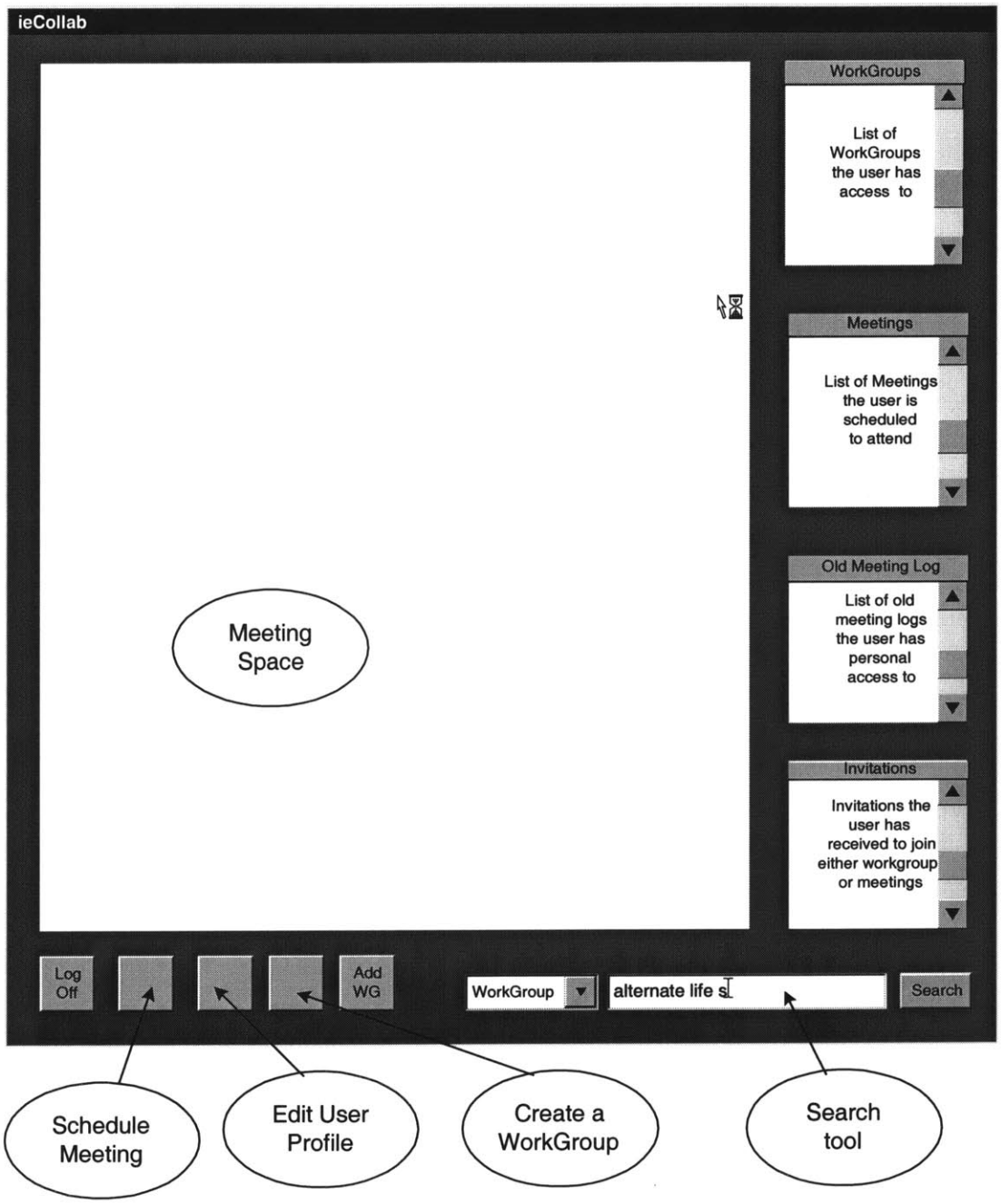


Figure 11: After Login

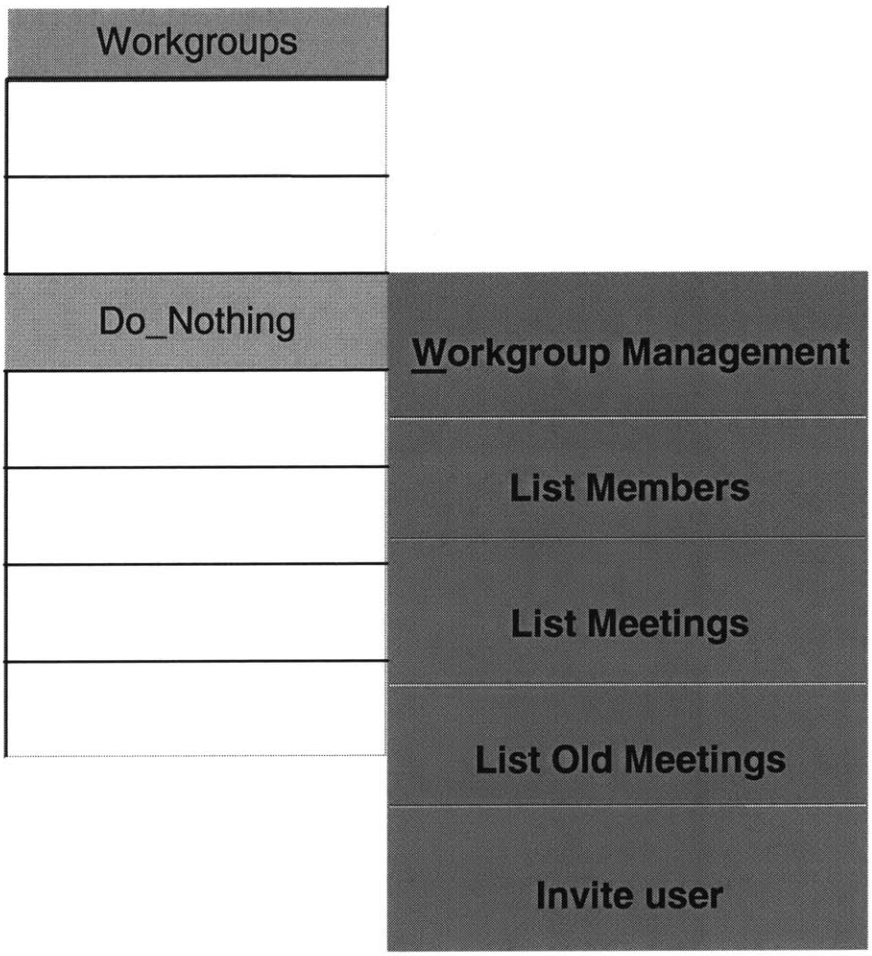


Figure 12: A Sample menu after a right click on a list item

Options on menus do not pop off separate windows. The pop up should be attached to its origin as shown above. This continues to the next level and so on. This is a recommendation so that we have a GUI which is logical in its construct.

Appendix B. Attributes of some System Elements

This appendix contains the definition of attributes for several elements of the system. The list is not exhaustive; it only provides some of the attributes of each element. More elements and attributes can be added, as the model (system) advances in the development process.

B.1 User Account

- User ID (Internal unique)
- Login name (Unique)
- Password

Note: The user could be identified in a unique way by using either the User ID and the Login Name (using an approach similar to the DNS - IP address resolution scheme).

B.2 User Profile

- Full Name
- Contact Address
- Workgroups the user is in
- Scheduled Meetings
- Invitations received for joining Workgroups and Meetings
- Privacy flag (if set, excludes user from searches)
- Preferences and settings. (Optional: allow customizable look and feel for the GUI)

B.3 Workgroup

- Workgroup ID
- List of members (includes users and *other Workgroups*)
- Workgroup Leader
- List of scheduled Meetings
- List of Meeting logs from previous Meetings
- The nature of the Workgroup (Private, Protected or Public).
    - Public: Visible in searches and anyone can join
    - Protected: Visible in searches but you need permission before you can join
    - Private: Invisible in searches and permission required before joining in

B.4 Meeting

- Meeting ID
- Scheduled Time
- Participants (includes users and invited Workgroups)
- Meeting Leader
- Meeting template which defines the Meeting structure
- Agenda

B.5 Meeting Templates

- Template Identifier

- Mapping between template role and Meeting participant
- The setting of the Meeting, or the corresponding Meeting interface
- The definition of roles

Two sample Meeting templates are described below:

B.5.1 *Free for all* Meeting

One participant is defined as the Meeting leader. However, she does not have any powers to revoke the membership of, or gag a user. *Free for all* does not mean that any user can join the Meeting, rather, it means that any user can talk without seeking permission. The power to limit the user list still rests with the person who schedules the Meeting. But she has an option to make the Meeting *free for all* users and Workgroups to join in. A round table might be an appropriate setting.

B.5.2 *Chaired* Meeting or Lecture

One person is designated as the chairman. He or she has totalitarian powers over the Meeting. Any user invited to such a Meeting can start the Meeting. However, such a Meeting does not get underway until the chairman joins the Meeting. The chairman can control the right to speak in the Meeting. Invited participants need not wait for the chairman's approval to join the Meeting but, they have to get his permission before they can speak. A rectangular table with the chairman at the head might be an appropriate setting.

Requirements Specification For ieCollab Version 2 Transaction Management

Specification Version 1.6

Requirement Analysis Team

Update from Version 1.5 by Bharath Krishnan, Alan Ng
Date: February 17, 2000
Participants on Modification:

- online Session: Alberto Morán
- offline Session: Bharath Krishnan, Alan Ng

References and Links
(All references are stored at http://collaborate.mit.edu/1.120.html)

• Requirements Specification v1.2 for ieCollab Version 1	January 22, 2000
• Requirements Specification v1.0 for ieCollab Version 2	December 9, 1999
• Requirements Specification v1.1 for ieCollab Version 2	December 22, 1999
• Requirements Specification v1.2 for ieCollab Version 2	January 17, 2000
• Requirements Specification v1.3 for ieCollab Version 2	January 17, 2000
• Requirements Specification v1.4 for ieCollab Version 2	January 18, 2000
• Requirements Specification v1.5 for ieCollab Version 2	February 2, 2000

Outline

1. Introduction

This draft presents the system requirements of ieCollab version 2. A separate document describes the system requirements for ieCollab Version1 [1]. This document specifies the functional requirements of ieCollab version 2 and how it relates to version 1 system requirements. The purpose of this document is to solicit input from other members of project to ensure that all parties agree on the system requirements.

1.1 ieCollab Versioning System

The ieCollab system will be developed in several phases/versions. Each version focuses on a specific functional component of the overall ieCollab system. The four versions/phases identified are:

- Meeting Management Environment (ieCollab Version 1):
  Allows distributed users to setup and manage online meetings. This includes functionality to keep track of meeting agenda, meeting participants, user profiles and meeting styles.

- Transaction Management (ieCollab Version 2):
  Allows the ieCollab server to track users' usage of ieCollab's meeting management services and charge fees on a per-transaction basis. Allows other A.S.P.'s to provide meeting management services to their clients transparently.

- Collaboration Server (ieCollab Version 3):
  Supports a set of interactive collaboration tools, such as chat tools, whiteboards, for group communications. Users should be able to pay for the use of these collaboration tools on a per-use basis as defined in the transaction management environment from Version 2.

- Application Server (ieCollab Version 4):
  Allows multiple distributed meeting participants to work on the same documents concurrently using third-party applications, such as CAD tools and spreadsheet applications. Users should be able to pay for these third-party software on a per-use basis.

This document specifies the system requirements for ieCollab version 2, focusing on the Transaction Management component of the system.

2. General Architecture

ieCollab version 2 will build upon the meeting management capabilities of the version 1. In addition, this version will have the capability to function as a transaction based meeting server. The architecture of which is briefly described below in Figure 1.

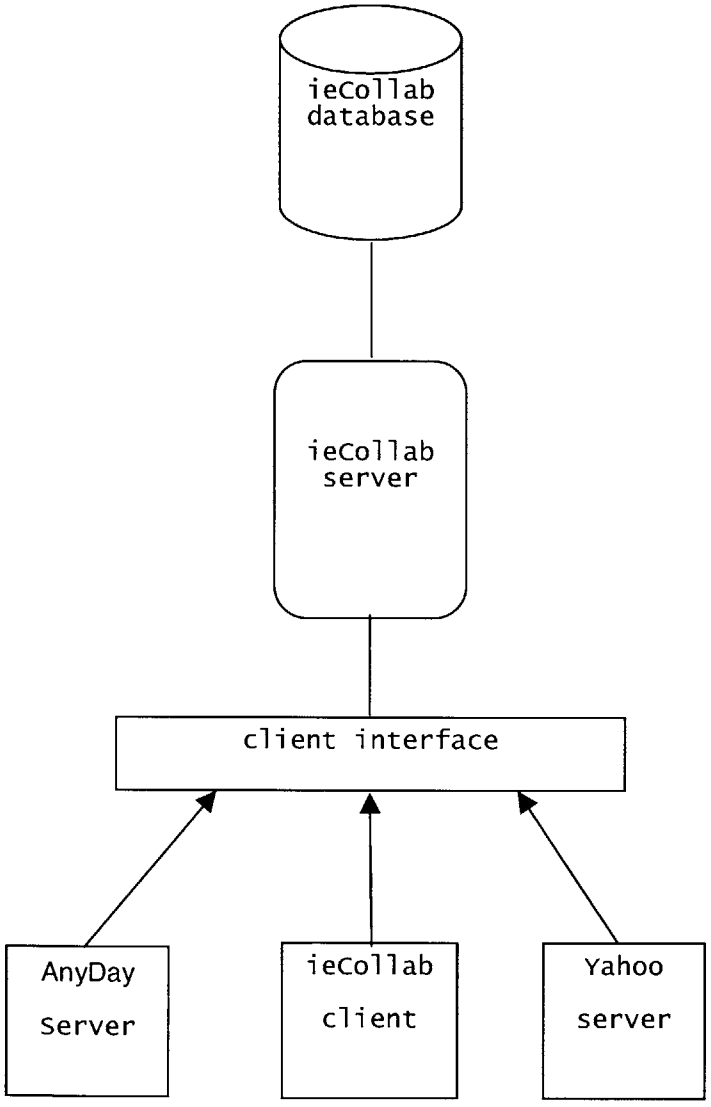


Figure 1: Architecture of icCollab version 2

ieCollab will provide transparent meeting management services for other Application Service Providers. Take for example a web calendar service like www.Anyday.com [2], which provides basic calendar and scheduling services on the web. ieCollab will offer to handle all the meeting management services for this A.S.P. in a transparent manner. Thus, the thin client interface to ieCollab is replaced by the other A.S.P's web interface. When a user logs on to a meeting through the A.S.P's portal, he will actually be connected to the ieCollab meeting server. This service is charged on a transaction basis. The ieCollab server will keep track of usage of its services. Users will then be billed for the time & applications they used.

3. System Specification

3.1 Overview

The use cases in this document focus on the transaction management component of the system. It is important to note that the use cases presented in this document focus on the use cases of ieCollab as a backend server, which provides transparent meeting management services to end users through another A.S.P. server, such as AnyDay.com or Yahoo.com [3]. The use cases in this document add to the use cases in the ieCollab Version 1 specification. The ieCollab will also be available through ieCollab's own thin client at the same time.

ieCollab Version 2 has the following four functional components (in addition to the functionality in Version 1):

- Set Service Pricing: allows the ieCollab Manager to define pricing schemes for ieCollab services.
- Create A.S.P. Account: allows the ieCollab Manager to establish an A.S.P. account with the ieCollab server.
- Create proxy account: allows an A.S.P. to establish an account on ieCollab on behalf of the A.S.P.'s end users transparently; i.e., the personal information of the user present on the A.S.P. is used by the A.S.P. to create the account.
- Track Service Usage: allows the ieCollab server to track usage of ieCollab services on a per-transaction basis (please see use case description on transaction management for more details).

Steps in use cases are grouped into two categories: steps in A.S.P. Server and steps in ieCollab server. Only steps in the ieCollab Server category are considered part of the ieCollab system. Although the A.S.P. Server is an external entity, we specify its corresponding steps in each use case to show how the end user's requests are mapped to ieCollab service requests through the A.S.P. server.

3.2 ieCollab System Entities

To facilitate explanation of the use cases, we describe the following ieCollab system entities and their relationships:

3.2.1 User: a user can be either a Normal User (as mentioned in Version 1 specification [1]) or an A.S.P. User who uses ieCollab via an A.S.P. server. Each user who has an ieCollab User Account is identified by a unique user login name. If the user opts for meeting services through an A.S.P., the A.S.P. will use the user's login name and other data in its database to create the ieCollab account for that user. A user has the following attributes:

- full name
- login name (unique)
- password
- broker identifier
- a list of workgroups the user is in
- a list of scheduled meetings
- a list of preferred meeting templates
- an account profile

The broker identifier specifies the A.S.P. server (such as Yahoo.com or AnyDay.COM) through which the user is using ieCollab. Each user has only one broker. The broker of a user has the right to retrieve and store user profile and meeting setup on behalf of the user.

3.2.2 A.S.P. Server: an A.S.P. Server who has an ieCollab A.S.P. Server Account is identified by a unique server login name. This account is used by the A.S.P. server manager. An A.S.P. server account has the following attributes:

- an A.S.P. name
- login name (unique)
- password
- an account profile

3.2.3 Account Profile: each account profile has the following:

- an account owner
- a pricing agreement
- payment method (usually credit card, could also be a monthly payment)
- billing address
- a list of transactions and invoices
- The GUI definition (When we allow each user a customizable user interface)

The account owner is either a User or an A.S.P. server. (i.e. the A.S.P. Server manager)

3.2.4 Meeting: each meeting has the following:

- a unique meeting ID
- a scheduled date/time
- default meeting template
- agenda
- security level
- A work group (As defined in the Version 1 specification [1])

For more details, please see version 1 specification [1].

3.2.5 Workgroup: A workgroup consists of a list of users and their roles in the group. Multiple meetings can use the same workgroup to define its participant list. Users of a workgroup are also called Workgroup Members. Each workgroup can have one Workgroup Leader and has a unique workgroup ID. For more details, please see section 2.4.2 of the Version 1 specification [1].

3.2.6 Transaction: for each transaction, ieCollab server will record the following:

- date/time of transaction
- principals involved
- usage

The principal of a transaction is the party that initiates the service request and is responsible for paying for the transaction. For example, if a user uses ieCollab directly without going through a 3rd party A.S.P., the user is recorded as the principal. If a 3rdparty A.S.P. server requests for ieCollab services on behalf of its users, the A.S.P. server can identify itself as the principal of the transaction.

Usage is defined by the types of services and quantities of services used. Quantity of a service can be defined by a combination of the following parameters: time duration of service, frequency of access, bytes transferred, bytes of information stored at the server, number of participants in a meeting. An I.S.P revenue model can be used. This includes a standard monthly subscription fee and an hourly charge. The exact rates for various services we offer in future can be decided by the sales management group.

3.2.7 Meeting Template: please see version 1 specification [1].

3.2.8 Meeting Agenda: please see version 1 specification [1].

3.3 Actors in Use Cases

3.3.1 A.S.P. Server Manager
This is the super-user of the A.S.P. server.

3.3.2 A.S.P. User
This is an end-user who is using the A.S.P. server. Although the user may use the ieCollab services through the A.S.P. server, the use of the ieCollab server is transparent to the end user.

3.3.3 ieCollab Manager
This is the ieCollab service manager, who sets usage and pricing policies.

3.4 Use Cases

3.4.1 Use Cases for ieCollab Manager

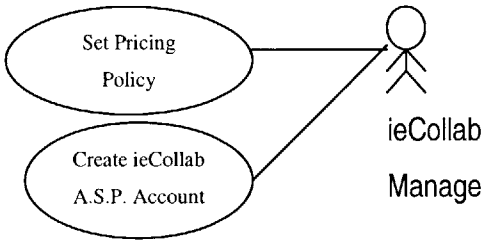


Figure 2: Use cases for ieCollab Manager

3.4.1.1 Setup Pricing Policy

This use case is started by the ieCollab Server Manager. It allows the manager to specify pricing policies for ieCollab services. For each service type, the ieCollab Server Manager can specify the unit definition for measuring service usage, and price per unit. The following service types are provided in Version 2: create/query/update user account and info, create/query/update meetings, create/query/update workgroups, and start/join meetings. As more features are added to ieCollab Version 3 and 4, more service types may be provided.

Service usage maybe measured in terms of:
- time duration of service
- frequency of access
- number of queries performed

- bytes transferred
- bytes of information stored
- number of participants in a meeting.

For each service type, the ieCollab Server Manager must specify which type of measurement will be used, and the price per unit of service usage.

3.4.1.2 Create ieCollab A.S.P. Account

An ieCollab Manager must setup an A.S.P. server account for the A.S.P. with ieCollab before the A.S.P. users can use the ieCollab services. To create an A.S.P. Account, the following information must be provided: name of A.S.P., login name, password, contact name/address, and billing information as specified by the contract between A.S.P. and ieCollab.

3.4.2 Use Cases for A.S.P. User

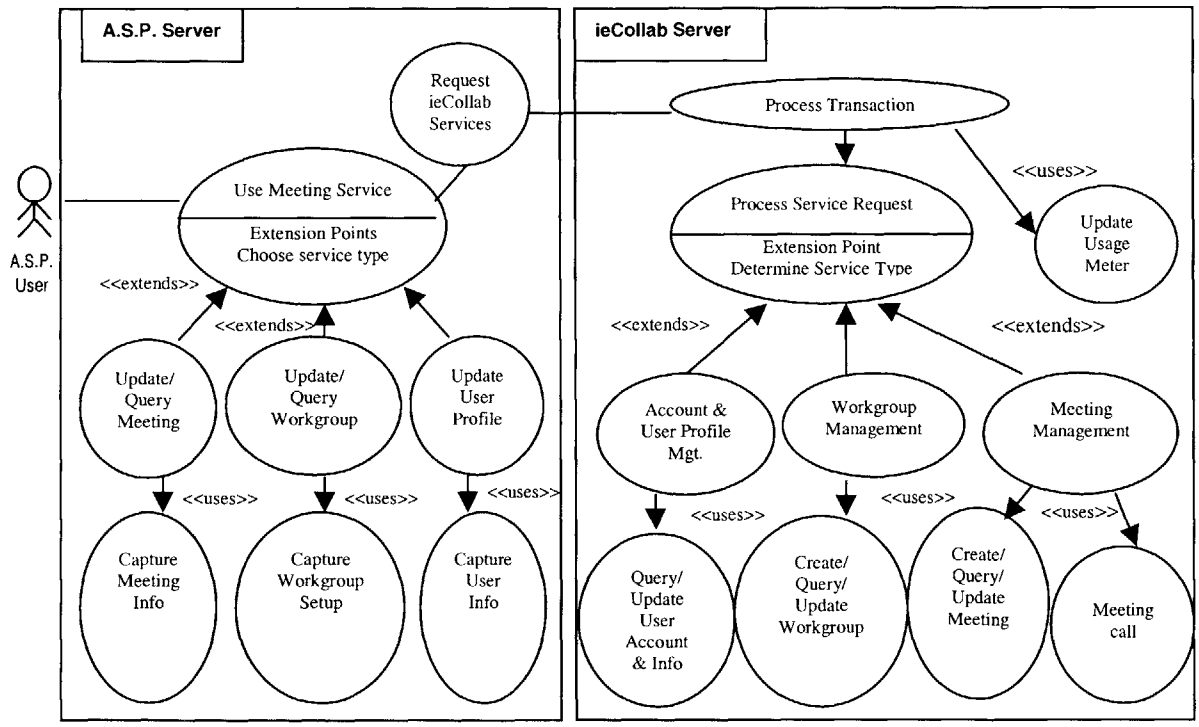


Figure 3a: Use Cases for A.S.P. User (Process Transaction)

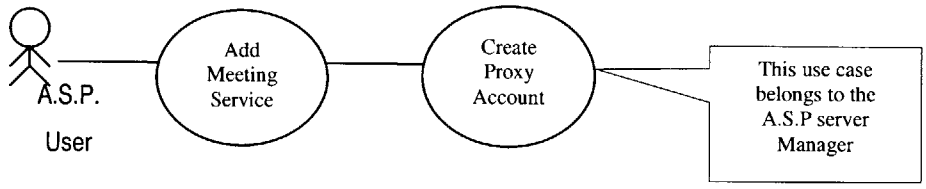


Figure 3b: Use Cases for A.S.P. User (Add Meeting Service)

3.4.2.1 Process Transaction

This use case is started by an A.S.P. User when requesting to use meeting service. The A.S.P. User's request will trigger the A.S.P server to send a service request to the ieCollab server. The service request will contain the A.S.P. user's login name and password, type of service requested, and other necessary data for completing the request. It is to be noted that the A.S.P. server acts on the behalf of the user. Any reference in this section to the A.S.P. server should be construed as the A.S.P. server acting on behalf of the user. The ieCollab server will process the service request as a transaction. The following steps are included in the Process Transaction use case:

- **Process Service Request:** There are several extended use cases based on the service types. The three main categories of sub use cases are: Account and User Profile Management, Workgroup Management, and Meeting Management. These are explained in section 3.4.2.1.1

- **Update Usage Meter:** Logging and tracking an A.S.P. server's or individual A.S.P. user's usage of ieCollab services

3.4.2.1.1 Sub Use Cases in Process Service Request

The following use cases are extended from the Process Service Request use case:

3.4.2.1.1.1 Account and User Profile Management: this use case includes Creating, Updating, and Querying of ieCollab User Accounts and User Profiles. ieCollab allows an external A.S.P. server to perform the following types of requests:

- **Query User Profile:** given a user login name, ieCollab server retrieves and returns that user's user profile.

- **Store User Profile:** given a user login name and a user profile, ieCollab server stores the user profile for that user.

3.4.2.2 Workgroup Management

This use case is triggered when the A.S.P user requests to Create/Modify Workgroup through the A.S.P server. This use case corresponds to the Workgroup Management use case in ieCollab Version 1 specification. Readers should refer to Version 1 specification [1] for details of this use case.

As an interface to its workgroup management, ieCollab allows an external A.S.P. server to perform the following types of requests:

- **Query Workgroup:** query workgroups based on workgroup IDs. This ID is a unique identifier for the workgroup.

- **Store Workgroup:** given a workgroup identifier, store a workgroup setup information. Please see section 3.2 for a detailed specification of a workgroup.

3.4.2.3 Meeting Management

This use case is triggered when the A.S.P user starts a Meeting Call through the A.S.P server. Note that ieCollab server expects the A.S.P server to capture meeting selection. This use case corresponds to the Meeting Management use case in ieCollab Version 1 specification. Readers should refer to Version 1 specification [1] for details of this use case.

ieCollab allows an external A.S.P. server to perform the following types of requests.

- **Query Ongoing Meetings:** this will cause ieCollab server to return a list of ongoing meetings currently at that ieCollab server.

- **Query Meeting:** given a meeting identifier, ieCollab server will return the meeting setup associated with that meeting. Please see section 3.2 for a detailed specification on the attributes of a meeting entity.

- **Store Meeting:** store meeting setup information under the specified meeting identifier. Please see section 3.2 for a detailed specification on the attributes of a meeting entity.

- **Start Meeting:** given a meeting specifier and a user ID, ieCollab will start a meeting using the meeting setup associated with the meeting identifier. All meeting state (current participants and their locations, meeting logs, communication channels among participants) will be maintained at ieCollab server.

- **Join Meeting:** given a meeting specifier and a user ID, ieCollab will connect the user to the specified meeting. An error status will be returned if no on-going meetings

correspond to the meeting specifier. An access denied message will be returned if the user ID is not permitted to join that meeting.

3.4.2.4 End Transaction

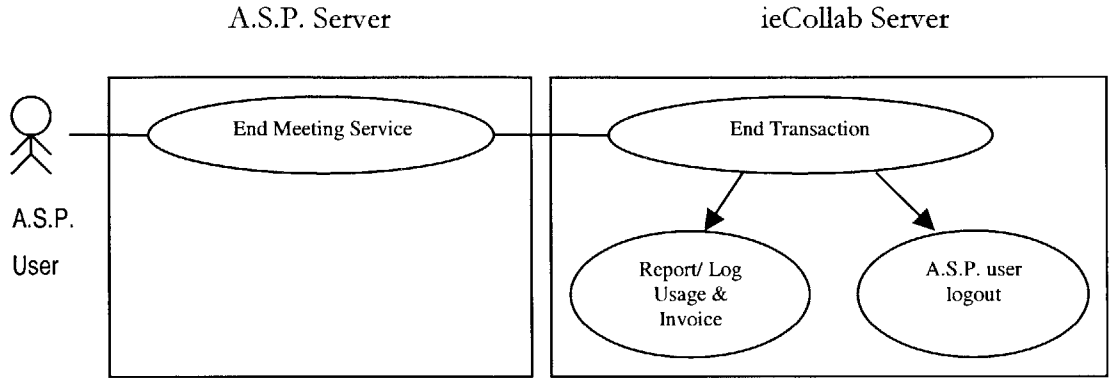


Figure 4: Use Cases for A.S.P. User (End Transaction)

The transaction is considered terminated when the A.S.P. user terminates the use of the ieCollab's meeting service. ieCollab will report the final service usage and invoice back to the A.S.P. server, and the A.S.P. server may decide to logout of the ieCollab server at the end of the transaction.

3.4.3 Use Cases for A.S.P. Server Manager

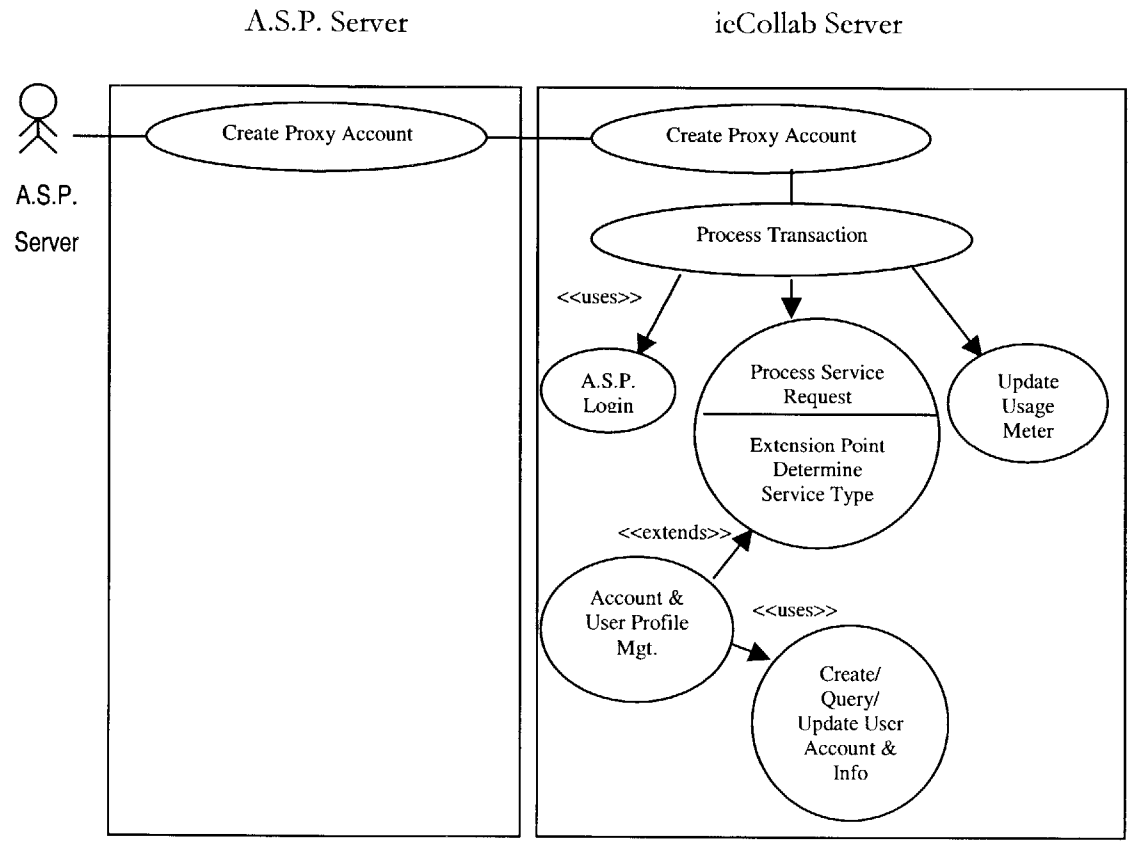


Figure 5: Use Cases for A.S.P. Server Manager (Create Proxy Account)

3.4.3.1 Create a proxy account

This use case is used by the A.S.P. server manager when the A.S.P. user opts for meeting management services. The process of creating an ieCollab account is automated by this use case. In other words, the A.S.P. uses the user profile for the user it has in its database to create an account for the user on the ieCollab server.

3.4.3.2 A.S.P. Login: An A.S.P. server can log itself in the ieCollab server by providing a valid ieCollab A.S.P. account name and password. Once an A.S.P. server logs in, it can request various ieCollab services, such as to create new user accounts, store/retrieve user profiles and meeting setups on behalf of its A.S.P. users. This step applies to A.S.P.'s which have a pre-established ieCollab A.S.P. accounts, which includes the pricing agreement and payment information.

3.4.3.3 Create User Account: An A.S.P. server can create an ieCollab user account on behalf of a user transparently. In order to do so, an A.S.P. server must log itself into ieCollab server first by providing a valid login name and password. After the A.S.P. server has been authenticated, it must then specify the user's name, login name, and selected password. In response, the ieCollab server must then create an ieCollab user account using the given information, and mark the A.S.P. server as the broker of the user.

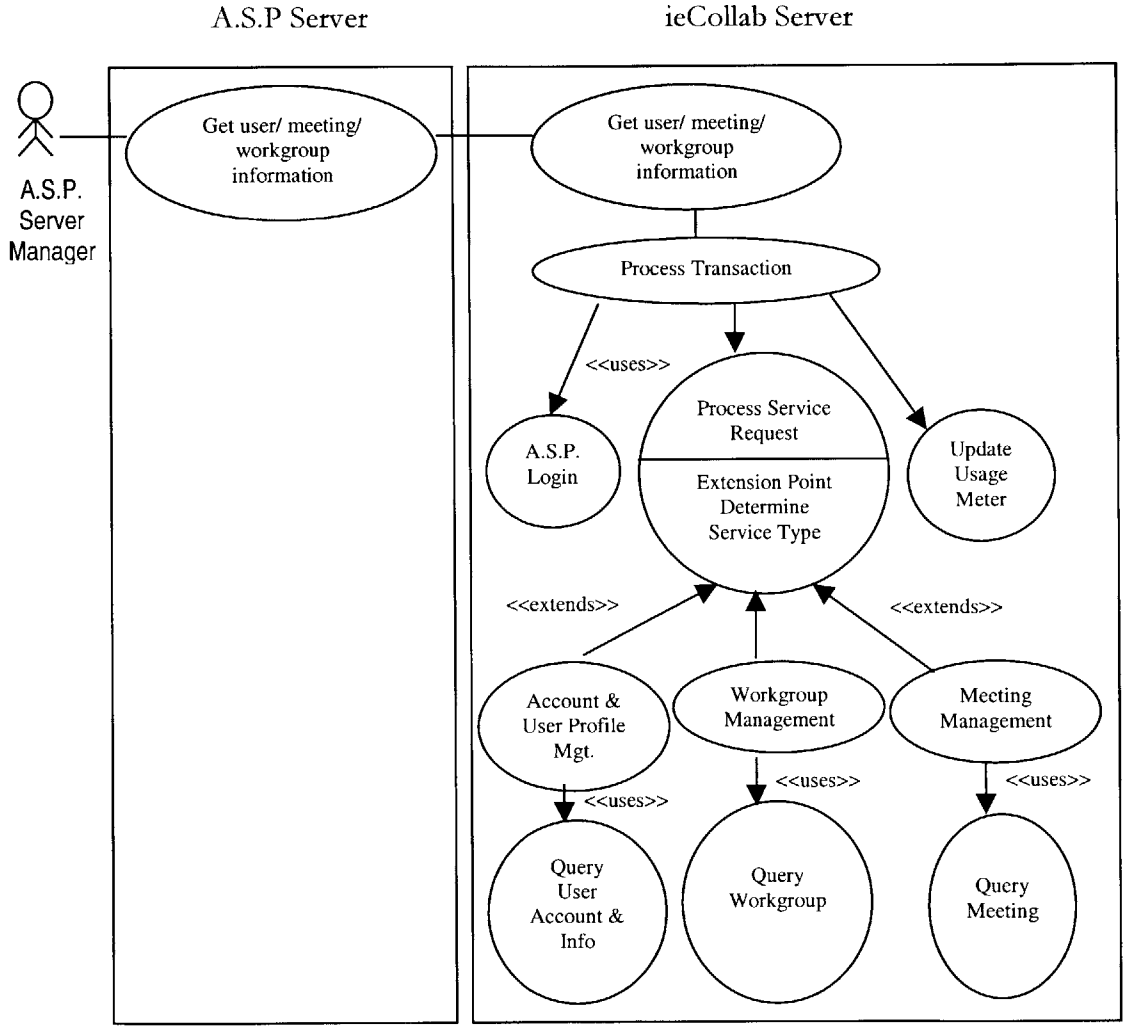


Figure 6: Use Case for A.S.P. Server Manager
(Get User/ Workgroup/ Meeting Information)

3.4.3.4 Get user/ workgroup/ meeting information

This use case is used by the A.S.P. server to get information from ieCollab about its users so that its own databases can be updated. An example would be an A.S.P which provides calendar services. The A.S.P Manager would then be able to automate the following task: Get meeting schedule information for users, update the calendar database so that the user's calendar would reflect the scheduled meeting.

4. References

[1] Moran A. "ieCollab Version 1 specification 1.4". 2000

[2] http://www.AnyDay.com February 3, 2000

[3] http://www.yahoo.com February 3, 2000