Managing a Distributed Software Engineering Team

by

Bob Yang

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degrees of

Bachelor of Science in Computer Science and Engineering

and

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1998

Author .....
Department of Electrical Engineering and Computer Science
May 19, 1998

Certified by .....
Feniosky Pena-Mora
Assistant Professor
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Managing a Distributed Software Engineering Team

by

Bob Yang

Submitted to the Department of Electrical Engineering and Computer Science
on May 19, 1998, in partial fulfillment of the
requirements for the degrees of
Bachelor of Science in Computer Science and Engineering
and
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Managing a software engineering project has traditionally been a challenging task due to ambiguous specifications, unpredictable implementation times, and last-minute bugs. For even the best project managers, there is no exact science to keeping a project on time and within budget. In the future, the task of managing a successful project becomes even more challenging as the complexity of software products increase. This document will address issues involved in managing a geographically distributed software team, an important area of complexity being introduced into software teams today. Key factors, such as team integration and resolution of cultural differences will be explored and methods of bridging these gaps will be discussed. It will be shown that explicit statements of expectations, similar to a list of rules for engagement are necessary for efficient communication. Building a level of trust and friendship between the distributed teams through working closely and communicating frequently is vital in the early phases of the process. The investigation will also show that assigning each team member a rigid role and a rigid set of responsibilities is not the best way to organize the team, but that each team member should feel responsible for contributing to all phases of the project.

Thesis Supervisor: Feniosky Pena-Mora
Title: Assistant Professor

Acknowledgments

I would like to thank Professor Feniosky Pena-Mora who read through countless iterations of this paper and provided encouragement and advice. Special thanks all the members of the DISEL team, Kareem Benjamin, Humberto Chavez, Juan Contreras, Gregorio Cruz, Siva Dirisala, Professor Jesus Favela, Juan Garcilazo, Lidia Gomez, Emily Hung, Karim Hussein, Sergio Infante, Pedro Ledesma, Felix Loera, Reuben Martinez, Rene Navarro, Charles Njendu, Josefina Rodriguez, Marcela Rodriguez, Simonetta Rodriguez, Diana Ruiz, Christine Su, and Tim Wuu, without whom this project would not have been possible. Finally, I would like to thank my parents for their support and patience over the past four years.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The purpose of the project manager in a team is to provide leadership to the members and ensure that the deliverables meet budget, schedule, and quality expectations. Traditionally, the project manager produces a project schedule, a resource budget, and a work plan. The project manager is also responsible for intangibles such as team morale, team harmony, and managing the expectations of the client. The project manager should provide guidance in decision making to ensure that the mission and vision of the project are adhered to. This means that during debates, all opinions should be heard, but once a decision is reached, the team should be fully committed to it. Half-hearted attempts by some team members can result in poor quality and schedule slippage as other members must pick up the slack. Division among the team is even more troublesome because team morale is adversely affected by a fragmented team.

Any problem should be resolved in a manner which minimizes the risk of project failure. This means making decisions on what features to include or cut, what bugs to fix immediately, and what bugs to postpone, so that the final product is ready on the delivery date. A balance must be maintained between meeting deadlines and improving quality and features.

During meetings, the project manager should make sure that the agendas are appropriate and that the discussions stay on track. In software engineering, the project manager must be able to take a step back from the low-level details and have

a vision of what the final product will look like, what market need it will fulfill, and who the end users will be.

1.1 Scheduling and Resource Allocation

The first challenge a project manager encounters is estimating how much time a project will require. In software engineering, there are several steps in the development process; requirements analysis, design, implementation, test, and maintenance [13]. One of the main factors influencing what methodology to adopt is the size of the software product. For example, there are very low project management requirements for a project with 1000 lines of code. Informal, ad hoc methods of design, implementation, and testing are sufficient. This model does not, however, scale up very well to medium and large systems (Figure 1-1) [13]. If 20 programmers were asked to create an air traffic control system, it is unlikely that they will produce anything usable or on time with an ad hoc approach. Instead, more formal methodologies must be used to manage cost, schedule, and quality.

1.1.1 Measuring the Scale of Systems

There is no universal definition for small, medium, or large scale systems. A variety of parameters, such as the number of modules, the number of functions, or the number of lines of code have all been used. Of these, the most commonly used one is lines of code (LOC) because there are several cost estimation methods based on this attribute. In general, a project with two thousand lines of delivered code (KDLOC) is considered small, whereas, 100 KDLOC is usually considered large (See Table 1.1)[13].

The management of different scale projects require different approaches. A small project may not require a lot of oversight and much of the time can be spent on programming. The system will probably have only a few modules, so the design phase can be relatively short. Formal reviews and walkthroughs do not need to be done as often as they are done in large projects because the team members will be constantly working with each other's code in a tightly knit group. The team is also

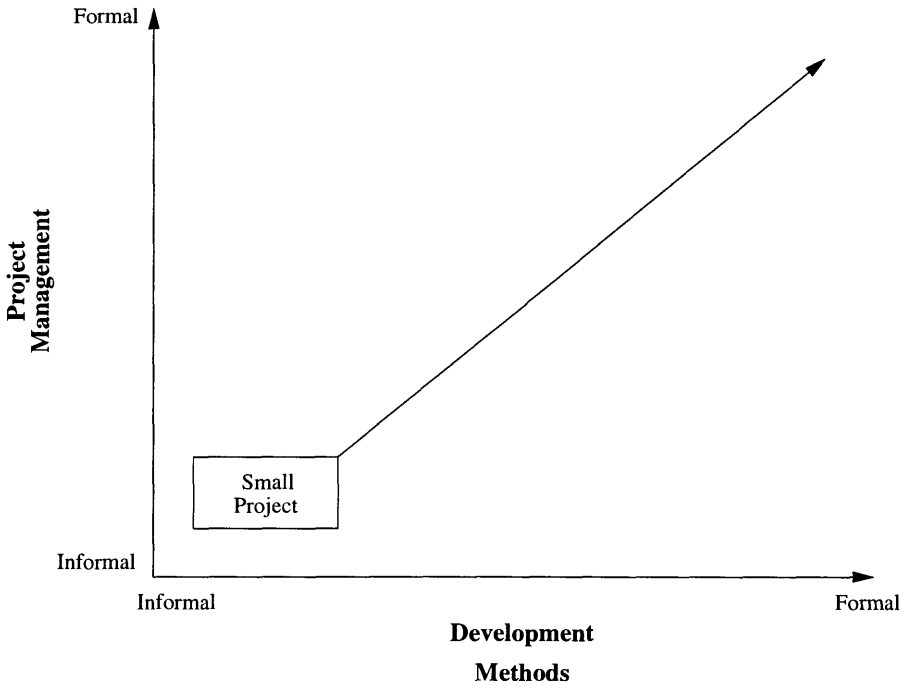

Figure 1-1: The Problem of Scale [13]

Program Size	Lines of Code
Small	2 KDLOC
Intermediate	8 KDLOC
Medium	32 KDLOC
Large	128 KDLOC

Table 1.1: Typical Program Size in KDLOC

much smaller, so the boundaries between areas of responsibility are much less well defined.

A very large project on the other hand, will require strong modularization and a longer design phase. It would be impossible for one person to track and understand all of the code and all of the interfaces between the modules. Frequent reviews and walkthroughs become necessary to ensure that the complex interactions between modules are managed properly and that interdependency issues are resolved as soon as they appear.

1.1.2 Cost Estimation

The goal of cost estimation is to identify key parameters which characterize the project, and use the value of these parameters to estimate the cost of the project. The cost of a project can depend on many parameters, but it is generally agreed upon that project size is the primary factor which controls project cost [13]. Some other parameters that may affect cost are programmer ability, project complexity, and reliability requirements [13]. For example, a real-time, mission critical system will cost more to develop than a similar-sized word processor because the first product is more complex and has performance and quality requirements much greater than the second product. In fact, reliability requirements generally increase the required effort exponentially rather than linearly because the effort required to detect and fix bugs becomes exponentially greater as the number of bugs decrease [6, 13].

A Simple Model

The most common approach to estimating cost is to make it a function of a single variable, the project size [13]. Basili proposes the following equation [2]:

$$EFFORT = a * SIZE^b$$

where a and b are constants. Values for these constants are determined by performing regression analysis on previous projects. Watson and Felix performed analysis on

more than 60 projects from IBM Federal Systems of sizes ranging from four KDLOC to 400 KDLOC. They found that a good estimating function was[17]:

$$EFFORT = 5.2 * KDLOC^{.91}$$

These constants came from analysis of projects by only one company. Different companies with different cultures and different methodologies will generate different constants. Different sized projects will also alter the cost estimation. A similar study conducted on smaller projects (two KDLOC) showed that effort varied linearly with size [2]:

$$EFFORT = a * KDLOC + b$$

This suggests that as a project grows beyond a small size, the resulting increase in complexity causes effort to grow exponentially.

COCOMO Method

A more complex model for estimating cost has been constructed by Boehm [3, 4]. This model called COnstructive COst MOdel, (COCOMO) takes into account many other factors, such as complexity, reliability requirements, and programmer capability. The basic model is composed of three steps:

1. Calculate the effort required using a simple cost model based on size ($E = a * (KDLOC)^b$).
2. Determine the values of a set of 15 multiplying factors from different attributes of the project (See Table 1.3).
3. Multiply the initial effort estimation by all the multiplying factors to obtain the final effort estimate.

In calculating the initial estimate, the values of a and b are determined by the project type. COCOMO classifies projects into three different types– organic, semidetached, and embedded. An organic project is one in which the development team has

System	a	b
Organic	3.2	1.05
Semidetached	3.0	1.12
Embedded	2.8	1.20

Table 1.2: Constants for different project types

had experience with. Examples include simple business applications and data processing systems. At the other end of the spectrum, an embedded project is one in which the performance and reliability requirements are highly stringent. Examples of these are satellite control systems and air-traffic control systems. Projects which fall in between, such as enterprise management software and database software are classified as semidetached. Each of these types of projects have different values for a and b (See Table 1.2).

For example, if the skill and experience of the programming team is judged to be exceptional, then this attribute will cause the overall cost estimation to be .70 of the nominal value derived from the simple cost model. All of the attribute factors are multiplied with each other and then with the initial estimate to arrive at the final estimate. This method takes into account project criteria that the simple cost model ignores. For more detail on the COCOMO method, refer to [3].

Accuracy of Estimates

The two preceding models operate on the principle of transfering cost estimation into a problem of size estimation. Although it may seem that this is only substituting one problem for another, size estimation has the advantage of being easily broken down. The size of a program is the sum of the sizes of its individual modules, whereas the cost of a program is most often not the sum of the costs of its modules. Thus, the estimation of size (KDLOC) is a more manageable problem and more accurately determined.

Even with advanced models, cost estimation is still only educated guesswork. The coefficients and attribute factors in COCOMO are determined thorough regression of

Cost Drivers	Rating				
	Very Low	Low	Nominal	High	Very High
Product Attributes					
Required Reliability	.75	.88	1.00	1.15	1.40
Database Size		.94	1.00	1.08	1.16
Product Complexity	.70	.85	1.00	1.15	1.30
Computer Attributes					
Execution Time Constraint			1.00	1.11	1.30
Storage Constraint			1.00	1.06	1.21
Virtual Machine Volatility		.87	1.00	1.15	1.30
Computer Turnaround Time		.87	1.00	1.07	1.15
Personnel Attributes					
Analyst Capability	1.46	1.19	1.00	.86	.71
Application Experience	1.29	1.13	1.00	.91	.82
Programmer Capability	1.42	1.17	1.00	.86	.70
Virtual Machine Experience	1.21	1.10	1.00	.90	
Programming Language Experience	1.14	1.07	1.00	.95	
Project Attributes					
Modern Programming Practices	1.24	1.10	1.00	.91	.82
Use of Software Tools	1.24	1.10	1.00	.91	.83
Development Schedule	1.23	1.08	1.00	1.04	1.10

Table 1.3: Multiplier factors for different project attributes

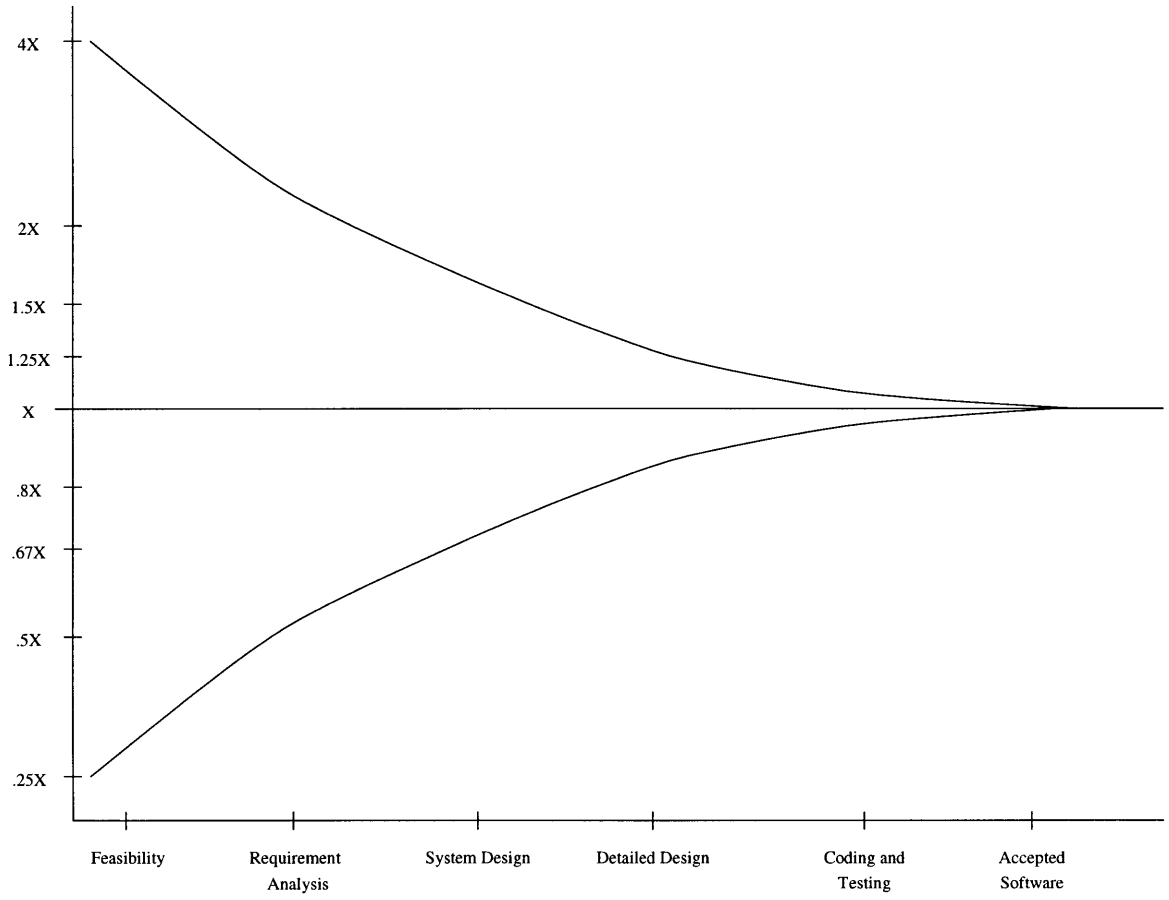

Figure 1-2: Accuracy of Cost Estimation [13]

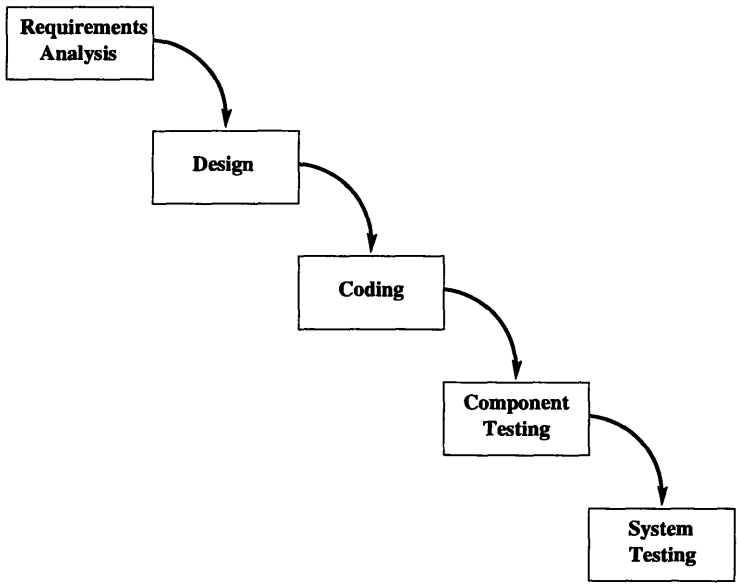

Figure 1-3: The Waterfall Method

past modules and may not apply well to new projects. The judgment of the estimator plays a strong role in the COCOMO method when attribute values are selected and will vary from person to person. As Figure 1-2 shows, the primary way cost estimates increase in accuracy is by actually monitoring progress as the project evolves.

1.2 Project Scheduling

There are may different ways of allocating project time amongst the various phases of the software lifecycle. Several different methodologies have been proposed, each with its own advantages and disadvantages.

1.2.1 Waterfall Method

The traditional model for planning the phases of a software development project uses the waterfall method (Figure 1-3) [6, 13]. This method calls for the linear, sequential execution of analysis, design, coding, component testing, and system testing. Each step is isolated from the others and ends with a deliverable that is certified by a quality control team. The teams are only told what inputs to expect and what outputs to

produce for each phase. The minimum deliverables in order are:

1. Requirements document
2. Project Plan
3. System Design Document
4. Detailed Design Document
5. Test Plan
6. Final Code
7. User Manuals
8. Project Review

This is the most widely used development method because it is conceptually very simple. It also covers all of the phases of a software project that must be done anyway. Proponents argue that completing the tasks in any other order will result in the end result being less successful [13]. This model also has the benefit of having a formal review of the products from each phase. The quality team must formally conduct a review of the products from each phase, measure them against the requirements outlined initially in the project plan and verify that all of the requirements are satisfied. Thus, at the completion of a phase of the product cycle, there is a guarantee of certain minimum quality standards.

However, Brooks argues that the flaw with this method is that it assumes defects will occur only in the implementation, and that the design and specification are good and realizable [6]. Should this not be the case, the cost to fix a flaw is very high and will most likely throw the project off schedule since changes have a ripple effect through the other phases of the project. Another drawback to this method is the assumption that requirements can be frozen and completely defined before the project proceeds. Often, it is the case that requirements may continually evolve as the project progresses or it may be desirable to develop a portion of the system before full requirements are complete. Most projects are not design-once processes, but a continual evolution of the design, given feedback from implementation and test. This

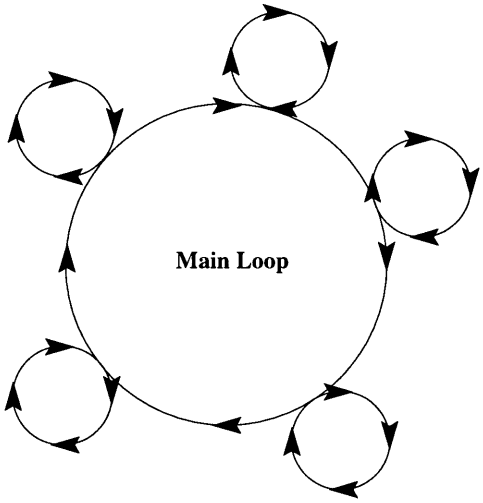

Figure 1-4: The Incremental Method

makes the incremental-build method (Figure 1-4) a much better candidate for use because requirements, design, implementation are incrementally advanced so that each iteration has the advantage of information and lessons learned from the previous iteration [6].

1.2.2 Incremental Build Method

The idea behind the incremental build method (Figure 1-4) is that once the high-level concepts are laid down, the implementors begin their work and constantly give feedback to help the designers flesh-out the lower level details. Thus, the team cycles through design, implementation, and test continuously. A good example of a commercial software company which uses this method is Microsoft. They instituted a policy of rebuilding an application every day. Every application undergoes a recompile at the end of each day, incorporating the new code checked in for that day. Usually, this will produce a functioning version of the program with new functionality and bug fixes. Thus, the team has a firm grasp of the status of the product every day and can see tangible evidence of progress towards completion.

The feeling of constantly being in touch with the status of the project is very important for a project manager [14]. At any given moment, he or she must know

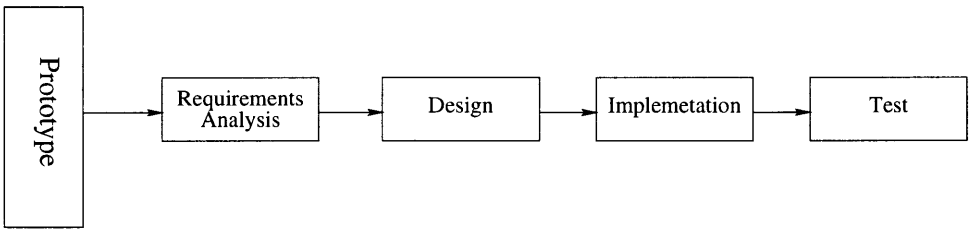

Figure 1-5: The Prototype Method

exactly what the status of the project is. Another benefit of this model is that there is always a working version available for immediate delivery. Demonstrations are easier to give because there is constantly a working, evolving prototype available. This model works well in sitations where the requirements are well-defined and the increments are very clear [13]. In a project with requirements that are unclear, it is very difficult to apply this model because the increments are equally unclear. If the first increment starts the project off in the wrong direction, fixing this can be very costly.

1.2.3 Prototype Method

The prototype method addresses the weakness of the waterfall method in situations where the requirements are amorphous and difficult to establish (Figure 1-5). It is a hypothesis driven model. The team hypothesizes a solution to the problem and builds a quick and simple prototype to as a proof-of-concept exercise. This prototype is then analyzed and the requirements are refined. The end result is a more stable set of requirements [13]. The development then proceeds in more or less linear fashion.

The danger and drawback behind using this model lies in the difficulty of establishing scope on the engineering prototype. The prototype is a throwaway product, so development must be rapid and cheap, yet sufficient to give the team a feel for the requirements. Balancing these goals can be a difficult task and may produce cost and schedule overruns if the prototype is too ambitious, but an underambitious prototype will lead to costly adjustments in the requirements document late in the project.

1.3 Allocating Project Time

In managing the development of the IBM 390 operating system, Frederick Brooks found that allocating 1/3 of the time to planning, 1/6 to coding, and 1/4 each to component testing and systems testing allowed for sufficient time to develop a robust and high quality final deliverable[6]. The rationale behind allocating so much time for design is that a proper design makes development much easier and contributes to a better overall deliverable. It is also less expensive to fix a design flaw early in the project. Testing is essential for delivering quality and solving any interface problems. Coding is the only controllable aspect of the project. If coding runs behind schedule, the scope of the project can be reduced.

The project manager must be careful when the initial schedule is created. False scheduling or schedules with vaguely defined deliverables result in a poor final product [5, 6]. Gantt and PERT diagrams must be created to show the critical path necessary for success. This will also encourage the team to continue to press forward even if one module is behind schedule. Sharp, well-defined milestones are a key to maintaining momentum.

1.4 Managing Scope

Throughout the life of a project, from inception to completion, a project manager must constantly maintain a balance between time, resources, and features. In most cases, the time and resources available for a project are fixed once cost estimation, the workplan, and the schedule have been completed. During the requirements phase, the team must agree upon a feasible feature list given the time and resources available. The project manager needs to make sure that assumptions for how long a particular feature will take is realistic. People tend to be overly optimistic when estimating the time necessary to complete a task [14]. The project manager should also prioritize each task according to importance and get the entire team to support the prioritizations [14].

Should some aspect of the schedule slip, the project manager has three options– to extend the schedule, to allocate more resources to the task, or to reduce the scope of the project. Extending the schedule may not be a possibility if there is a hard deadline or if the product is trying to time the market correctly. In any case, too many schedule slippages can create a sense of inevitable failure in the team and may even cause the product to become obsolete before completion [6, 14].

At the other end of the spectrum, the team may come up with new features that will improve the product, or a change to an existing design or requirement can arise. A formal process must be established to control these change proposals to the original specification. A Software Configuration Manager (SCM) is a role which is formally responsible for evaluating change proposals [13]. The IEEE defines *software configuration management* as "the process of identifying and defining the items in the system, controlling the change of these items throughout their life cycle, recording and reporting the status of items and change requests, and verifying the completeness and correctness of items" [12]. The SCM takes the change proposal, elicits feedback from the team on the consequences of incorporating the change, and either accepts it, or rejects it. If the change is accepted, then resources and responsibility will be assigned by the project manager to implement the change.

An analysis team is responsible for defining requirements during the requirements phase, and a design team is responsible for design during the design phase, but often requirements and design will change in other phases, such as the implementation phase. In this situation, it is the job of the SCM to control these changes because the development process is usually only able to handle changes in code, and not changes in requirements and design [13].

1.5 Maintaining Quality

In order to deliver a high quality final product, quality must be constantly monitored. The later an error is found, the greater the cost of fixing it [6, 13]. At the beginning of the project a quality assurance plan should be made that specifies the tasks which

need to be performed during various phases. These tasks are usually reviews and audits which objectively evaluate a deliverable [13].

1.5.1 Validation and Verification

The job of verification is to determine whether or not the products of a given phase of software development fulfill the requirements and properties defined for that phase. The job of validation is to determine whether the final software product fulfills the initial requirements determined during requirements analysis [13]. This process is different from the test phase of a project, when test engineers will perform functional and structural testing.

Validation and verification is a continuous process which spans all phases of the project because errors occur not only during coding, but in requirements gathering and design as well [6, 13]. The main tools for verification are walkthroughs and inspections.

The software inspection process was started by IBM in 1972 to improve software quality and increase productivity. An inspection is a formal review of a product by peers. The purpose of an inspection is to carefully scrutinize a product for defects. This close scrutiny has the added benefit of ensuring modularity, clarity, simplicity, and adherence to standards [9].

The review process involves peers reviewing a product privately and then together as a review group to detect defects and potential defects [10, 11, 13]. During the review meeting, a moderator sets the agenda and maintains control of the meeting. The moderator should set an entry criteria, i.e. what specific types of defects the group will focus on, and ensure that everyone enters the meeting having closely reviewed the product beforehand. The meeting will then proceed with the reviewers raising issues, the team discussing the issues, and the author either accepting an issue as an error or explaining why it is not an error. The meeting should *not* be the time to suggest fixes to the error, as this may make the author defensive. At the end of the meeting, a list of open issues and resolved issues should be published. The project manager must be careful to ensure that it is the product which is being reviewed,

and not the author. This is important so that open, honest opinions can be voiced without triggering defense mechanisms of the product's creator.

In addition to increasing the quality of the system, these reviews also provide the project manager with useful information for project monitoring. If consistently high error counts are found, then the project may need to be adjusted for a longer test period, or the design may need to be re-examined. The project manager should schedule these reviews into the project plan at the project's inception as a critical process for quality control.

1.6 Team Organization

A project team can be organized in many ways. Usually, the team will be divided into sub-teams, each with a specialized task and its own set of objectives, mission, division of labor, and interface definitions. Allocating work within these sub-teams requires careful planning so that work is neither duplicated or omitted. Harlan Mills proposes the "surgeon model" [15]. In this model, there is one chief surgeon, who is responsible for all aspects of the work required of the team. The rest of the team do everything they can to support this chief surgeon. For example, in a programming team, one person is responsible for the code, while the others document the code, test it, control revisions, and share ideas with the chief coder. This allows for a high degree of conceptual integrity. One mind is in charge of all the modules, interfaces, and implementation. There will be no confusion in evaluating effects of code changes. The disadvantage to this is that the model does not scale well. Once the team grows to greater than five members, the marginal contribution from each additional member is very low.

It is difficult to keep everyone in the project involved during all phases. For example, there is very little to do for the developers during the requirements gathering phase. It is important to keep everyone continuously involved in the project or else the team will lose cohesion and the individual member will lose interest and cease to contribute.

1.7 Project Documentation

Having a document or a set of documents which explicitly state a project's objectives, specifications, budget, schedule, and organization chart is critical to the success of a project. When these documents are formally created, informal thoughts and discussions are organized in a manner which will clearly expose any gaps and inconsistencies which would have slipped through otherwise. These documents also help a team maintain its focus and eliminates any chance of incorrectly communicating expectations.

A good project workbook will contain all documents, objectives, specifications, technical standards, and administrative memorandum. It is critical that this workbook is constantly updated with the latest version of every document. This creates a central repository to which the entire team can refer for the latest developments and progress.

All decisions made during each phase of a project should be documented. This way, decisions can be justified later and all assumptions will be clearly outlined. Results of reviews, minutes from meetings, and logs of open and resolved issues may seem trivial at first, but keeping these documents well organized will aid the members in staying abreast of the project and provide justification in case decisions need to be re-examined later in the software configuration management process.

1.8 Design and Implementation

Brooks holds conceptual integrity to be the most important priority in designing a software system [6]. Conceptual integrity means adhering to one interpretation of the objectives which the product will meet. When a large group is involved in the design phase, there will be many interpretations, resulting in poor realization of all the interpretations. There will be features that do not integrate well together and features which are unrelated to the original objectives. Brooks suggests that there should be one or a small group of like-minded individuals who design the entire system [6]. In

essence, the design process is, and should be, a rule by tyranny, not by democracy. This creates one coherent vision for what the final product should be, instead of a compromise of different ideas. A large committee will also take much longer to reach a consensus on major issues.

There is also the question of how much knowledge to expose to the programmers. Should each programmer know the code for every other module in the project? David Parnas asserts that this could lead to problems because programmers will start making assumptions about code that is not their own [16]. He suggests completely isolating a programmer from modules that that are not his or her own. This forces a good design with formal, well-defined interfaces.

During the design process, there are two main approaches normally taken, the top-down approach, and the bottom-up approach [13]. The top-down approach involves moving from the abstract to the specific, in terms of features and modules. The bottom-up approach focuses on building up, from the specific to the general, so all the functions would be defined first, and then grouped into modules. For a more detailed treatment of the design process, refer to [7, 18].

1.9 Communication

A project manager's primary task in managing daily operations is to facilitate communication among team members and to monitor the status of the project. The project manager should not be tempted to act upon every problem that arises, but to allow the group members to fix easy problems. He or she should focus more on the big picture and head off long-term problems. It is vital that regular meetings are held and that the project workbook is kept current. These are the two most effective ways to avoid a misassumption [6].

Any small appeals or open issues which are non-critical should be noted and acted upon during special sessions which Brooks calls "Supreme Court Sessions" [5]. At these sessions, the items are aired to the entire group and a decision is made, with the head of the role responsible for the item having the final say. These sessions avoid

costly delays and help maintain the momentum of the project.

If a team is unfamiliar with the protocol to follow in meetings and communications, a team contract is a good tool for establishing basic rules of engagement [1]. A team contract is a document which lists a set of established behavioral guidelines such as how soon to expect a response to an email, how to air grievances, and how decisions which affect the whole team will be made. The contract is created by soliciting opinions from the entire team on an issue and discussing it until a consensus is reached. Once the rules have been established, each member agrees to abide by them for the duration of the project. Consequences for infractions should also be established by the team. Sapient Corporation, a software consulting firm, found high success with a one dollar monetary fine for each infraction. Once the penalty was introduced, infractions dropped dramatically.

1.10 Challenges in a Distributed Environment

The task of a project manager in a distributed environment is even more difficult than in a traditional environment. In addition to managing day-to-day operations, he or she must also worry about the synchronization between the different teams and the level of communication between the various members. Each member must be made to feel a sense of ownership of the project early in the process if their efforts are to be maximized. It is especially difficult to establish a feeling of cohesiveness if there is a geographic barrier separating team members who have never met previously. Traditional team building activities, such as team lunches or dinners are not usable. Traditional teams also have the opportunity to interact socially throughout the course of a day through random casual conversations so that at the end of the first few weeks, a certain level of familiarity is reached. These types of integration paths are severely hampered by the communication barriers introduced by geographic separation. Coordinated team activities are difficult because the available communication mediums are so limited. Video conference technology allows for face-to-face meetings between large groups of people, but it does not convey enough details, such as facial

expressions and body language, to become a viable substitute for face-to-face meetings. It is acceptable for pure information transfer between two parties, but when it is used for social communication, especially between groups of people, its limitations significantly detract from the experience because there are so many more components to social group interactions than voice and facial expressions. It is especially difficult if the team members have never met in person. One cannot assess the personalities and abilities of the team members without having extended contact with them and observing them work and interact with peers.

Aside from team integration concerns, decision making and managing day-to-day activities are more complex because feedback from all team members must be received, processed, and responded to before an issue is laid to rest. There can be a significant lag time between a proposition and a resolution because spontaneous group meetings are difficult to arrange and trying to establish consensus using email is a futile process.

Another major issue which arises from geographical distribution is team culture. In the United States, people from the West Coast are generally reputed to be more relaxed and laid back than their East Coast counterparts. This can create friction if a project manager has a management style which conflicts with the culture of the sub-teams. This situation is exacerbated if teams are located in different countries, when customs and language issues also come into play. For example, a culture focused on consensus building and mutual reinforcement will clash with a team that has an entrepreneural, fast-moving culture. One team will feel left-behind and unappreciated while the other will feel hindered and unnecessarily delayed.

Chapter 2

The DISEL Project

This project modified and applied the traditional tools and techniques of project management to an environment in which team members were geographically distributed. The project team, named DISEL (DIstributed Software Engineering Lab), consisted of seven students at MIT and ten students at CICESE, an institute of higher learning in Mexico. Together, these two groups created a software tool to improve distance learning. The team operated using a small company paradigm. Each member was assigned a particular role and was expected to learn and perform the duties and responsibilities of his/her role. The time frame for this project was one academic year (nine months). A distributed environment presented new challenges in overcoming communications limitations, cultural differences, and language difficulties. Key issues such as document synchronization, setting rules of engagement, and avoiding miscommunication due to cultural differences are challenges that are an order of magnitude more complex in a distributed setting than in a centralized setting. Old methodologies required modification in order to remain effective in addressing these challenges. This research will focus solely on the management aspects of the DISEL project. Other aspects, such as design and implementation difficulties are beyond the scope of this document.

2.1 The Team

The DISEL team was originally comprised of seven members from MIT in the United States, and ten members from CICESE, a research institute in Mexico. Each member of the team was assigned a well-defined role which he or she assumed for the duration of the project. The roles were:

Project Manager Responsible for providing overall leadership to the team. He or she will create the project schedule, work plan, and monitor progress.

Analyst Responsible for documenting and assessing the requirements of the client. This requirements document provides the purpose behind the software product and supplies a framework upon which the designers can build.

Designer Responsible for designing a software product which satisfies the client requirements. A design document containing feature lists, usage scenarios, and object diagrams is generated. This document should be detailed enough so that a team of programmers may implement the requirements set by the customer.

Programmer Responsible for implementation of the software product, converting the design document into a tangible product.

Test Engineer Responsible for discovering software defects and providing test scripts for each module as well as the system as a whole.

Quality Control Engineer Responsible for ensuring that the correct process is followed for all aspects of the software lifecycle. Under the waterfall method, the Quality Control engineer must approve the final deliverable of each phase before the next phase can begin.

Validation and Verification Acts as the client's advocate, ensuring that the client's requirements are kept in focus. Responsible for creating the agenda to meetings and moderating the meetings.

Documentation Specialist Responsible for taking minutes and maintaining all documents generated in the project.

Software Configuration Manager Responsible for identifying and defining items in the system, controlling the change of these items throughout their life cycle, and verifying the completeness and correctness of these items [12].

Maintenance Engineer Responsible for maintaining the software product against external changes, requests by users, and re-engineering the product for future expansion.

All of the roles, with the exception of Documentation, Software Configuration, and Maintenance had more than one person responsible for it. Within these roles, one member was the head of the role, and the others were assistants (Figure 2-1).

Due to the fact that the DISEL project is an academic class, students could elect to depart the class freely during most of the project. This maps to workers leaving a company in a commercial environment. Through the course of the project, two members of the DISEL team left the project for various reasons, and one member took a three month leave of absence. The impact of these departures will be discussed later.

2.2 DISEL Objectives

The goal of the DISEL project was twofold. The first and primary goal was to educate the team members on the issues and complexities involved in medium-scale software development in a distributed environment. The second goal was to create a software product which can be used to address issues found in geographically distributed working groups. These goals were to be attained by immersing the team in a distributed development environment so that the members experienced it first hand. The software product to be developed was done in this distributed environment so that at the end of the project, both goals will have been accomplished.

2.3 The DISEL Environment

The academic setting of the project created an environment which was different from a commercial company in very important ways. First and foremost was that the members were not committed to the project for 40 hours per week the way professionals are. The team members pursued studies in other subjects in addition to this, and were only expected to spend about 10 hours per week on the class.

Secondly, there was instructional material to be presented by instructors, so the class formally met twice per week in a classroom setting (Figure 2-2). One of these

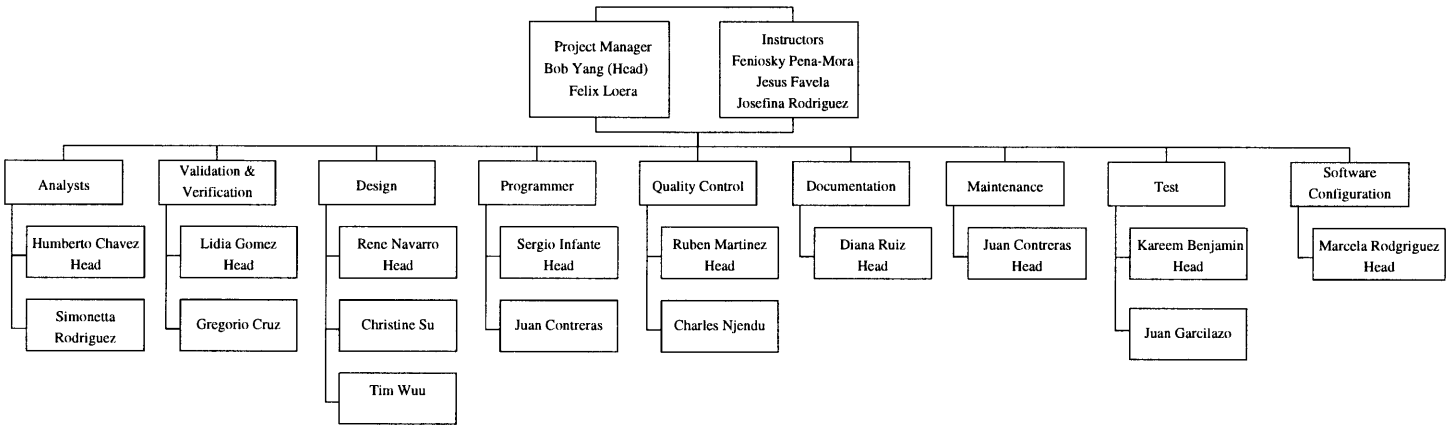

Figure 2-1: DISEL Organizational Chart at Project Start

meetings consisted strictly of lectures by the instructors, while the other one was a laboratory session when the entire team could meet to work on the project. These limitations in meeting time and student resources strained the project schedule towards the end, and were necessarily altered. The lecture sessions were eliminated so that more lab time could be made available to the class. Meeting twice per week helped keep the team updated on progress and moved things along faster.

To communicate between the MIT and CICESE portions of the class, several Internet-based tools were used. The telephone was not a primary medium of communication because the group wanted to test and push the envelope in the latest communications technology. Video conferencing was used instead during class sessions. A video camera set up on each side would broadcast images of the classroom to the other team. InPerson, a videoconferencing program ran on a Silicon Graphics workstation and displayed video from both the local classroom and the remote classroom side-by-side. These two video feeds were then projected onto a projector screen via an overhead projector.

Audio was transmitted using a module of Microsoft Netmeeting. This program was also used to transfer real-time written messages across during class sessions. There was only one instance of Netmeeting running, and usually four microphones which were passed around amongst the class members. Occasionally there was only one microphone, but a lack of microphones for everyone was not a major hindrance. The students all sat around one large meeting table with the video conferencing projection on one wall, and a Netscape window projected on another. A special program called WebPresent allowed the Netscape window at MIT to synchronize with the Netscape window at CICESE. This was useful for presentations involving slides and other visuals. It enabled the presenter to control the visuals in both locations.

Outside of class, email and Internet talk were the most common modes of communication. There was a project-wide mailing list established as an efficient way of making announcements to the all the members of the project. All of the emails to this mailing list were archived on the project web site. This web site also hosted the project documentation and provided a discussion forum which allowed for a thread

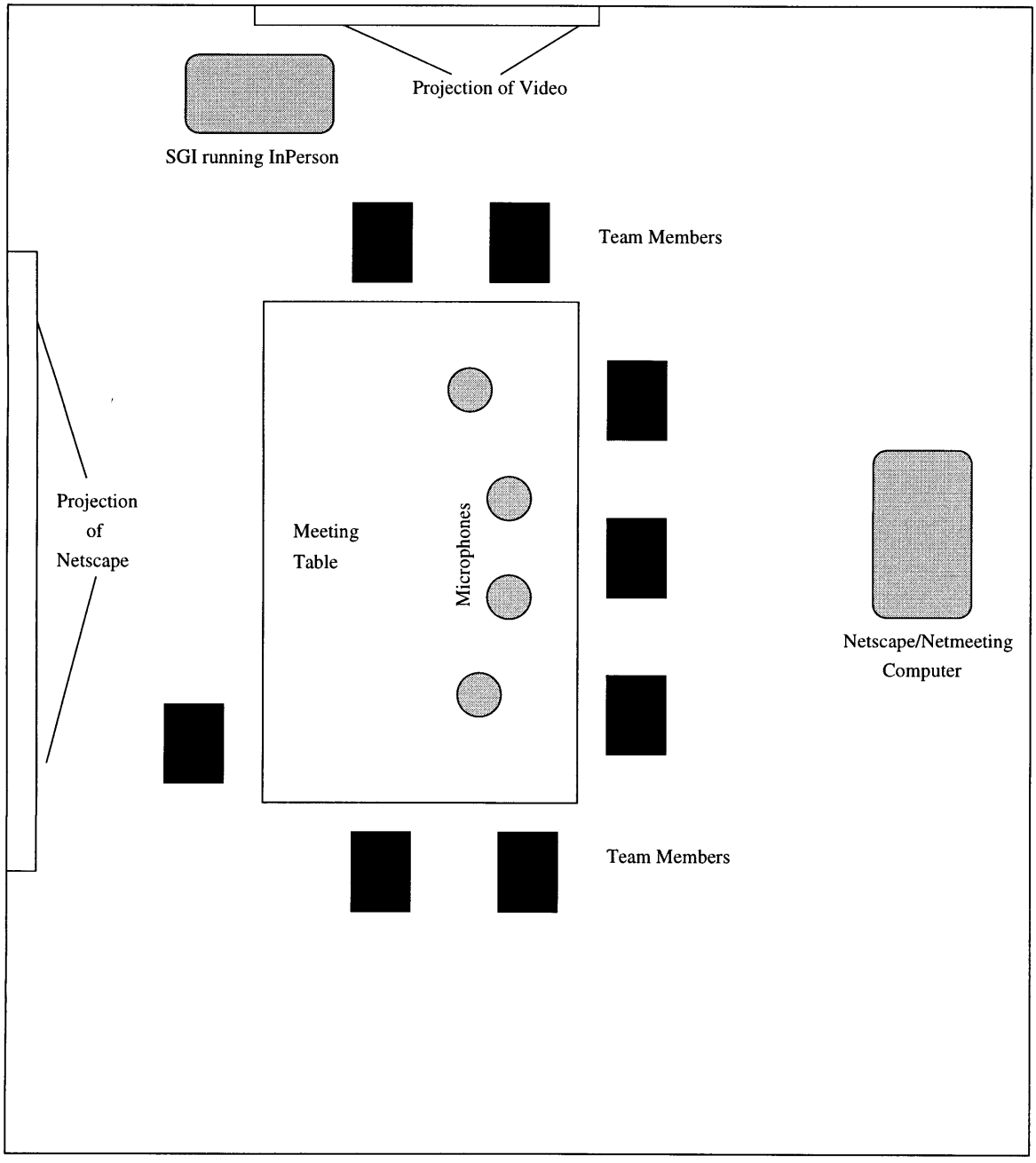

Figure 2-2: MIT Classroom Environment

based message posting system.

Chapter 3

Managing the DISEL Project

All aspects of managing the DISEL project fell upon the shoulders of two student project managers, one from MIT, and one from CICESE. The MIT project manager was the head of the project management team and the CICESE project manager was the assistant. The instructors played an advisory role in the project. They provided advice and recommendations, but the ultimate decision was made by the project managers. The project managers were responsible for creating the project schedule, setting the agenda for team meetings, setting action items to be assign to team members, and maintaining good communication between members of the entire team and within each of the sub-teams.

3.1 Limitations of the Environment

The fact that the DISEL project was conducted in an academic environment instead of a professional environment resulted in distinct differences in approach from the project managers. The project managers were limited in the authority they could exercise and in the time commitment they could demand from the team members. These limitations will be discussed in more detail in the sections that follow.

3.1.1 Team Hierarchy

The organizational structure of the team was established by the instructors before the start of the project (Figure 2-1). When the project started, each person was immediately assigned to a role within the team, depending on his or her skills. Usually, each role was the responsibility of a team of two or three, one from MIT and one from CICESE, with one person assigned as the role head, and the other assisting that person. This pre-selection of team hierarchy had the benefit of imposing structure on the team immediately and having someone responsible for each of the major tasks. This avoided a lengthy discussion on team organization and who will be responsible for what tasks. Given the inexperience of the student members, there would have been a high probability that a necessary task far down in the development process was missed during the initial definition of roles and assignment of responsibilities if the students were to define the team organization and tasks each person was responsible for. Any shift or redefinition of roles and responsibilities months into the project would have be distracting at minimum and potentially disastrous. It would have caused confusion among the members and increase the probability of an important task falling through the cracks.

The drawbacks to predefined roles are twofold. First, the team members do not understand the rationale behind the decision to use a particular team hierarchy. The members are not made aware of other options, and the benefits and drawbacks of each of these options. Second, the personalities of the team members are not taken into account in the assignment of roles. There was no time for the DISEL members to get to know each other and communicate for any length of time before the role assignments were made. This resulted in a few cases of personality conflicts between team members assigned to a role. This problem is not just a limitation of the organizational process, however, but of the distributed nature of the project as well. There is a high probability that even if the assignment of roles were postponed for a few weeks, there still would not have been sufficient interaction between the MIT members and the CICESE members. It was observed that the members of the two geographical

sites did not integrate closely until they were required to produce a deliverable. This problem of team integration will be discussed later in the document.

The basic process for establishing team organization was sound, but a few modifications may address some of the drawbacks to the process. First, time should be spent educating the team members on the different organizational options available so that they have a more complete understanding of the benefits and drawbacks associated with each of these options, as well as why a particular one was selected. The students should also be given a more in-depth overview of what each of the roles is responsible for. The role-based structure of the team was good in that it allowed each student to thoroughly understand a particular aspect of the software development process. In the first few weeks, however, there was confusion among the team members, especially on the MIT side because of the learn-your-role-as-you-go approach used. Team members did not understand what was expected of them and the exact responsibilities of their role. For example, the validation and verification engineer did not know that he was responsible for enforcing the agenda and moderating team meetings. In depth lectures were given on the responsibilities of each of the roles, but these lectures were not completed until one-third of the project time had elapsed. In the future, it would be beneficial to spend the first two weeks familiarizing the team members with more of the theory behind a software engineering team and the responsibilities of each role.

The problem of personality conflicts is a more difficult one to resolve because its resolution depends on overcoming the communication barriers associated with a distributed team. The main issue here is that the team members did not have a chance to interact beforehand in an manner similar to how they would interact during the actual project: working together to produce a deliverable. Having the team members participate in various group exercises before the assignment of roles can help the members make a more informed decision of who they want to work with, a factor that was not taken into account in the assignment of roles. The requirement that each role be composed of members from different geographic locations should be maintained as a way of promoting integration between the two difference geographic sites. Should this requirement be relaxed, the temptation to do over-the-wall engineering, where

one side completes a deliverable, and passes it off to the other, will be very high. This type of interaction would result in two very separate teams, instead of one unified team, defeating the purpose of the project. In the future, an improvement on the role assignment process may be to allow each team member to select one or two counterparts from the other geographic site, and have the entire group decide what role it would like to assume. Though this may not be common in companies, project groups in an academic environment are often formed this way. If one desires to preserve the real company feel of the project, then the groups should have a period of time before the project officially starts to complete some small projects together. A quick design exercise would be a good candidate for a small project.

3.1.2 Resource Limitations

The fact that the entire team was composed of students who had commitments to other classes resulted in a resource allocation problem because the time frame of a task is necessarily longer due to the fewer number of job-hours available per week. A task which a professional team is able to complete in one week would require the DISEL team approximately four weeks because each member of a professional team can commit 40 hours per week, whereas a student can only be expected to commit 10 hours per week due to other classes and activities. This resulted in a project plan that was approximately four times longer than a comparable project plan in a professional environment. This stretching of tasks created the danger of having team members lose focus and lose interest as they wait for a certain phase of the project to develop. For example, most of the team had very little to do during the requirements analysis of the phase, since they had no specification of features, no design, and no knowledge of the programming language and tools to be used. This idleness caused the team members to distance themselves from the project since they felt no sense of ownership. The team members not directly responsible for requirements gathering merely waited for the analyst team to complete the requirements document before they involved themselves in the project. Despite a call from the analysts to comment on preliminary drafts of the requirements document, the team was unresponsive and

no member took any sort of initiative in further exploring a topic of discussion. Members were merely content to go along with whatever the analysts stated. It was hoped by the program managers that each team member would be able to contribute to the requirements document. Discussion threads were started and drafts were posted on the web, inviting comment, but comments were few and unspecific. Even direct assignments to team members were largely ignored, or resulted in a half-hearted product. The project managers then lost their own sense of direction and purpose and were merely content to let the project drift off in a random direction of its own.

The example cited above was during the early stages of the project and can be partially attributed to poor team integration and expectation setting. The root of the problem, however, lies in the fact that a majority of the team was idle for an extended period of time during the analysis phase. In a professional team, a period of one week devoted to requirements gathering is not a significant length of time in a nine month project, but one professional work week is equal to about four DISEL weeks. A one month period of idleness can cause even the most determined team member to lose interest. A noteworthy observation is that the analysts were quite active during the design portion of the project, after their main responsibilities were complete. Similarly, the designers were quite active in assisting the programmers during the implementation phase. From these observations, it would appear that once a role team produced an important deliverable for the project, its members felt a sense of ownership in the project and were willing to commit time and effort into making the project succeed.

The lesson to be learned here is that a project manager should place a high priority on requiring each team member to play a significant role in a deliverable early in the project. One possibility would be to delay assigning roles to students until the requirements phase is complete. There will be one person nominally in charge of the requirements analysis. All of the other team members are assigned specific tasks/areas of investigation by this head analyst. Using this method, every member of the team will have played a major part in producing the requirements document, creating a sense of project ownership which is vital in motivating the

team. The drawback to this idea lies in the fact that this does not scale up to large teams. Requirements gathering would be an exercise in chaos if the team grew larger than 15 people. An alternate method would be to use the prototype model of project scheduling. The rapid development cycle of the prototype will result in a final product very quickly and will heavily involved all of the team members. This may be suitable for a professional environment, but in an academic environment, the definition of rapid is several weeks, an unacceptably long period of time.

The best solution is one in which everyone has a responsibility for the requirements, but a small group is responsible for maintaining the conceptual integrity of the project. Everyone on the team contributes their ideas for the requirements and submits them to a small subgroup who then weigh all the suggestions and conducts a review of them and publishes the final requirements document. This way, everyone is encouraged to think about the project and feel that they have the opportunity to contribute to it.

3.1.3 Authority Limitations

A project manager needs to have a certain level of authority over the team. A good project manager will work with team members when delegating tasks and responsibilities, but he or she must be able to make assignments and expect them to be carried out. In the DISEL project, the project managers were students. Outside of the classroom, the project manager was a student peer of the other team members. The general attitude of the project managers was one of trying to persuade and build consensus. There was an absence of the attitude of authority. This may seem to be beneficial at first, since the team members may seem more empowered, but there are times when a position of authority is necessary to hold people responsible for their assignments. For example, in most of the team meetings, it took a long time for an action item to be approved because the project manager must negotiate with the team member and explain the needs of the project. In one meeting, however, one of the instructors took charge and simply assigned people tasks based on need, with no negotiation. The attitude of the instructor was "I expect you to..." whereas the

attitude of the student project managers is "Could you please..." There is a distinct difference psychologically in those two attitudes which was felt by everyone on the team.

If a team member did not produce a deliverable on time, or produced one of poor quality, there was no recourse for the project manager, other than to re-adjust the schedule. The project manager had no way of evaluating any of the team members, and also had no way of receiving feedback on his performance. The problem with this lack of reward and penalty is that the project manager has difficulty in enforcing assignments and ensuring deliverables are of high quality. This is a flawed system because there is no compensation, be it monetary or in the form of a grade, explicitly tied to performance. The lesson to be learned from this experience is twofold. First, the project manager must be perceived as having a mandate of authority that carries beyond the classroom setting. This implies that one of the instructors must be the overall project manager, with a student serving as an assistant. This gives the position of project manager a legitimacy of authority that would otherwise be absent. A project manager with authority can formally evaluate a team member's performance, providing that team member with more motivation to excel in the performance of his or her tasks. The project manager should also consider giving monthly surveys and evaluations to each member so that the team members can get feedback on performance as well as make suggestions on areas in which the overall team can improve.

An alternative to having the instructors act as project managers is to create a new post, vice president, and have the instructor play that role. The vice president can conduct periodic checks of all the deliverables that the team produces and evaluate them. That way, the team members, though managed by project managers, are ultimately evaluated by the vice president.

3.1.4 Limitations of Technology

The DISEL project attempted to use the cutting edge of technology in executing the project. Traditional mediums for communication, such as telephone and postal mail

were abandoned in favor of a new breed of tools which take advantage of the ever expanding communication network known as the Internet. Due to the fact that this technology is still relatively immature compared to established methods of communication, the tools which the team used left much to be desired in certain cases. Indeed, it is the goal of the group to improve these tools to enable future distributed teams to interact more easily, and to establish a higher level of communication, conveying facial expressions and body language in addition to mere speech.

During team-wide meetings held every Tuesday and Thursday, there were two main tools used to communicate between MIT and CICESE. One tool, Microsoft Netmeeting, was used to convey audio between the two groups. Another tool, InPerson was used to convey video. Netmeeting is a tool which provides the capability of allowing various users on various computers across a network to come together and have an interactive "chat session." In these chat sessions, there is a window which displays text messages from the meeting participants, along with identification of the originator of the text message. Any participant may send a message to this chat window, but the message will be seen by all of the meeting participants. There is no facility for sending a private message to one specific participate, other than to open another Netmeeting session with only that participate attending. The audio portion of Netmeeting broadcasts audio from one location to another.

It was the intent of the DISEL team to rely primarily on the audio functionality of Netmeeting to communicate. In practice, the effectiveness of the audio left much to be desired. First, audio requires a significant, uninterrupted data stream in order to be intelligible to the other geographic team. Since the Internet is a public, shared resource, the available bandwidth between MIT and CICESE often times was insufficient to support the audio stream, so that speech was garbled. A significant portion of meetings were devoted to repeating sentences, often multiple times. Team members would often query if the other side could hear what was being said before making a point, and then reconfirming that the point was heard after he or she was done talking. Occasionally, after many repetitions, the speaker either gave up, or resorted to using text to send the message across. The quality of audio also placed a

limitation on the possible times in which the two teams could meet. Originally, the meetings were schedule for Tuesdays and Thursdays at 5pm EST. The CICESE team had issue with that time. Attempts to move meetings to an earlier time were initially unsuccessful because the audio quality between noon and 5pm EST was so poor that almost no coherent speech could get through. Finally, in January, the meeting time was moved to 10am EST, a time when the audio improved to the point where most statements did not need to be repeated. It is probable that with advancements in technology such as IPv6 and IP-multicast, the bandwidth problem will abate, but a method must be established whereby the team members can signal that audio was not heard clearly. Prefacing every comment with "can you hear me" is a waste of time and unacceptable. The signal to repeat does not need to be complicated, a circle on screen which flashes red when someone from another location desires the speaker to repeat himself is sufficient.

The video portion of the group meetings ran on a Silicon Graphics workstation and displayed an image of the local site and an image of the remote site, side by side. The video input was taken from a video camera setup at each site. Video suffered even more severely than audio from the bandwidth problems of using the Internet. Often, the frames were jerky or displayed nothing at all. The camera set up in the meeting room was usually not zoomed in far enough to be able to convey the individual faces of the meeting participates. It was impossible from looking at the video feed to establish who was present at the meeting, let alone the facial expression of the speaker. It was generally agreed by the team members that the video aspect of the meetings added little value, and was largely ignored.

This lack of visual contact between team members had a profound impact not only on the social interaction of the team members, but on the approach a project manager must take at meetings. It is common, in a face-to-face meeting for the manager to go around the table asking for status updates from each team. In this environment, there is no table, and the project manager finds himself speaking to an amorphous entity 2500 miles away. There is also no way for the project manager to see the expression on someone's face. Subtle signals, such as body language and facial expression can

sometimes convey more information than the spoken word. Often, a team member will be reluctant to voice opinions, especially if it goes against the group opinion. It not uncommon for someone to say one thing and mean another. If a team member agrees to something with a frown on his face, it may be an indication that he has a disagreement with the point, and the project manager should delve deeper.

In this situation, the project manager had no way of ascertaining facial expression at all. It is critical that in the future, a remote-conferencing tool be developed which allows facial expression and body language to be conveyed in addition to audio. A method which allows the participants of a meeting to be distinctly identified is a requirement as well. It is very disconcerting to address a team member, and then be told that he or she is not in attendance. The speaker's train of thought is disturbed and he or she must make a mental note to remind him or herself to speak to the person later. Even at the end of the project, most of the team members have no idea what their counterparts at the other geographic site looked like. Being able to picture someone while speaking to them is an intangible that is not missed until it is not there. It is significant to note that in the place of facial recognition, it became easier at the end of the project for students to recognize each other's voices.

As a consequence to this, the project manager should take role before all the meetings to verify attendance and try to create a nurturing atmosphere in the group to reduce the reluctance of team members to voice their opinions. This will also aid in identifying and familiarizing the team members with each other. No suggestion should ever be casually dismissed without careful consideration and any criticisms should be moderated so that only the idea is criticized, and not the originator of the idea.

Outside of the classroom, the primary tool used for communication was Internet talk. Again, this communication medium sets limits on the depth and frequency of communication. During talk sessions, no body language or facial expressions can be conveyed. The closest it comes to transmitting emotion is through the use of emoticons such as :-) for smile. Talk sessions are also not spontaneous. Appointments must be scheduled before a talk session can occur. This proves to be a problem when

information is needed immediately. One cannot just walk down the hallway to another office, or pick up the phone and call someone. The implications of these limits on social interaction will be discussed later.

Another software tool used for collaboration was a web-based discussion thread manager. Team members could post a message containing a topic for discussion and other members could post follow-up messages. Unfortunately, this tool never caught on. Initially, there were bugs and the instructions were in Spanish. Even after translation and debugging, it still was not a popular tool. The project managers attempted to start several discussions and specifically asked all members to post messages, but very few team members did. The failure can be attributed to several causes; first, many of the team members were reluctant to come out openly with their ideas or opinions, second, many were not interested in the topics discussed because it did not fall under their umbrella of responsibility, third, effective use of discussion threads requires that members check the web page often, something that never happened, and fourth, the project managers did not have enough authority to enforce participation and incorporation of the discussion threads into everyone's daily routine. If a project manger truly wants to use discussion threads, he or she must incorporate checking the web site into the daily routines of all the team members.

The final communications tool used is perhaps the most ubiquitous in cyberspace today, email. There was an email list containing of all the team members. Emails sent to this list were logged on the project web site so that important announcements may be recorded. Early in the project, the use of subject tags was adopted to help manage the large volumes of mail. [FYI] in the subject line meant that the message required no action on the part of the reader, whereas [Immediate Action] required a response in 24 hours. Email was mainly used to communicate announcements, pass along URLs of published project documents, and to set up appointments for talk sessions. Email will continue to be an important tool in distributed teams, but care should be taken not to flood the list with irrelevant emails. One of the most difficult challenges facing a project manager is to make sure the team members read and remember everything sent to the mailing list. The temptation to erase bulk mail is great. Care must be

taken to make emails short and relevant.

3.2 Team Dynamics and Integration

In the opinion of the project mangers, the first phase of the project was to allow the members of the team to familiarize themselves with each other. This should happen even before the project plan and project goals are established because of the nature of the work environment. None of the team members had known each other before the start of the project. This would not be a problem if the members would be running into each other and interacting for many hours on a daily basis, but in this situation, the team members will only be interacting with each other for a few hours each week. Thus, efforts to integrate the team became much more challenging. To complicate matters further, the geographic separation between MIT and CICESE meant that the two sides would never meet each other face-to-face. All of the team building efforts must take place in cyberspace. There was no chance that members from the two sides would run in to each other in the hallways, or chat with each other informally in the dormitory.

A solution to this lack of contact and familiarity is to have set lab sessions each week with no agenda at all. Team members simply come in to work on the project and make themselves available for communication. No meetings have to take place and nothing formal needs to be done. The team members only have to be present together so that if someone needs to ask something, they can have a chat session, or a video conference. The difference between this and email is that this communication will be interactive and instantaneous. Email usually has a lag time of a few hours before a response is received. This lab session makes communication easier and more frequent because the team members no longer have to negotiate for a time to have a chat meeting.

In a professional environment, each member of a team comes to work for at least six hours each day. In these hours, the person may not be communicating with other team members all of the time, but he or she knows that the other team members are

available for immediate conversation should the need arise. The scheduled weekly lab sessions would simulate this type of environment.

Distance, however, is not the only barrier a distributed team must contend with. The international nature of the teams creates language and cultural barriers as well. A project manager must distinguish between a language miscommunication, a cultural miscommunication, or a miscommunication due to lack of communication medium. It is a highly difficult task to isolate the cause of miscommunications, and often there is not one single cause.

3.2.1 The Language Barrier

All of the meetings and communications in the DISEL project were in English. This posed no problem for the team members at MIT, but for the members in CICESE, it was a difficult adjustment. English was not their native tongue and there were clear reservations towards impromptu conversations in English. The project web site was originally in Spanish, but was translated into English by the CICESE team. This was a challenging task for them. Throughout the first phase of the project, the CICESE team members were clearly hesitant to express themselves in English and lacked confidence in working with their MIT counterparts, fearing that their counterparts would find them unintelligent if they spoke in broken English. In one conversation with the validation and verification engineers, it was revealed that the CICESE V&V engineer would prepare everything that she would say during meetings a few days in advance. It was clearly evident that the CICESE team was uncomfortable with spontaneously expressing themselves in English. In reality, their English was perfectly understandable. Despite reassurances from the MIT team, this pattern of behavior continued for quite some time before a sufficient comfort level was reached so that the CICESE team became more comfortable expressing themselves. It seems that once a working relationship was established between the members of the two sides, the sensitivity to language lessened. The MIT team was told early on to try to simplify their vocabulary in an effort to make the CICESE team more at ease. This advice was not particularly adhered to, but there were no complaints from the CICESE

team. The CICESE team only understood 60%-80% of most conversations during meetings, but the CICESE team members never stopped a meeting because they did not understand a point. The project manager must be very careful to ensure that important points are understood by the entire team.

One method for ensuring that everyone understands the main points from a meeting is to post the salient points made at a meeting on the web. This has worked well in this project. The documentation specialist published a list of minutes from each meeting, along with agreements and action items. This allowed all of the team to refresh their memories or to review points that were not well understood. In the last quarter of the project, the quality control team also began to publish their points on the web. This practice of publishing everything may seem tedious and may make life difficult for the documentation team, but it is vital in ensuring that the entire team is on the same wavelength, no matter what language problems crop up.

As for the problem of lack of confidence in speaking ability, establishing a comfort level with one's peers seems to be the best way to overcome it. After a period of close and frequent interaction, even if solely on a professional level, it seems that the language problem was overcome. The interaction between project managers clearly demonstrate this. In the first few chat sessions, the CICESE project manager typed very slowly and haltingly, in an attempt to use correct English. He also stated that many of the CICESE team members had a lot of ideas in class, but were afraid to share them because they could not effectively communicate them in English. After repeated reassurances that the MIT team would not look down on them in any way if they didn't speak with perfect English, it was observed that more and more comments were made by the CICESE team members. Talk sessions also went much smoother since the CICESE team was less reluctant to attempt to express themselves in English. The language problem can be managed through constant encouragement and establishing trust, but it can be altogether avoided if language fluency is used as a criteria for selecting geographic sites.

3.2.2 The Cultural Barrier

The two institutions which make up the DISEL team, MIT and CICESE, have very different cultures. MIT has an atmosphere of individuality and entrepreneurism. Students are expected to perform tasks and make quick decisions on their own. It is not an environment where praise is expected, nor given very often. There is simply an implicit acknowledgment of a job well done when a task is completed, since it is expected that one will complete a task which one is responsible for. There is also a free and casual attitude towards decision making. Decisions are made quickly and work is begun immediately. There is not a lot of debate once a decision is made. Students here are also very assertive and outspoken. It is not common to withhold one's honest opinion. The drawback to this is that sometimes opinions can be blunt and may offend someone not used to the culture.

The environment at CICESE is more structured and group oriented. They reinforce each other's work with support and praise, and seek to build a consensus among the entire team before proceeding with a decision. There is also a reluctance for an individual to take-off on his own and place responsibility on himself. They are very respectful of authority and are not as outspoken as their MIT counterparts. For example, one of the designers in CICESE had a criticism of the other [MIT] designer's work. He did not directly tell the other designer, but instead asked the CICESE project manager to relay the information to the MIT project manager to handle. If the roles were reversed, it can almost be guaranteed that the MIT designer would have directly talked to the CICESE designer.

These cultural differences created several problems at the beginning of the project. The first crisis arose when the CICESE team felt that their efforts were unappreciated by the MIT members. They had translated much of the web site from Spanish to English, but a key part, the discussion threads, was still in Spanish. The MIT team focused on the need to translate this remaining part while the CICESE team expected positive feedback and support from the MIT team. As a result, a rift grew between the two teams. Responses to email came slower and slower and the spirit of cooperation

evaporated. Finally, after two weeks of this, a discussion to "air out" all the feelings was held. The MIT team had no idea how the CICESE team felt. There was honest appreciation for the hard work that went into translation, but the culture of MIT was such that the focus was on the next task rather than the work just completed. After this incident, there was a heightened sensitivity to the issue and subsequent critiques of completed tasks were always prefaced with encouraging remarks to meet the expectations of the CICESE team. This soothed the conflict and communications returned to normal.

Soon after this conflict, the MIT team took issue with a cultural gap. The agendas for each of the meeting sessions were pre-determined and very rigid. There was a set time period for presentations and discussions, with each person allotted approximately two minutes to make comments. These time limits were strictly adhered to. The MIT team felt that this format for meetings was too restrictive because it did not allow for the free exchange of ideas. Once a person spoke, he or she no longer had the opportunity to speak again. The CICESE team was accustomed to this strict rigidity of agenda, but the MIT team felt severely limited in their ability to express themselves. The MIT members were accustomed to lively, free-form debates instead of strict, moderated discussions. There are advantages and disadvantages to both approaches. The structured approach is not as supportive of spontaneous idea generation, but it also prevents one or two people from monopolizing the discussion time. A more free-form approach to meetings may allow more spontaneous discussions, but these discussions may throw the meeting off-topic and cause time over-runs so that other important issues are not addressed. Once the project managers were aware of this, a hybrid approach was taken in establishing meeting agendas. There continued to be a set time period for discussion, but within these discussion periods, anyone may speak at any time. This compromise worked well and put both sides more at ease in meetings.

A project manager must recognize that cultural differences are inevitable between groups from different geographic areas. He or she must try to manage these differences and monitor team interaction closely so that these cultural differences do not fester,

but are brought into the open. Compromise is usually fairly straightforward once the different viewpoints are communicated. Deeper personality biases, however, are more difficult to address. For example, some CICESE team members still would not present their ideas during meetings, even after working with the MIT team for six months. They instead preferred to make their ideas know offline to the project manager. This is just something a project manager will have to become accustomed to and learn to work with.

3.2.3 The Technological Barrier

The final major barrier faced by distributed groups is the technology barrier. The main limitation is the ability of team members to communicate with each other. The main communication tool used in this project, email was not instantaneous enough for quick exchanges of ideas. There were often times when a team member working on a task required knowledge from another teammate at that instant. With email, the worker is at the mercy of waiting for his teammate to check mail. There is no way to actively page the teammate for a communication session. This type of communication lag significantly decreases worker productivity. This type of delay plagued the DISEL project constantly. Waiting for technology to advance is one solution to the problem, but technological progress is unpredictable. In the meantime, the project manager should set aside a period of time during the day when everyone agrees to be available for communication. This way, emails can be responded to immediately, or talk sessions can be requested knowing that the other person is available.

The other technological barriers, such as the inability to convey facial expressions and body language, and the inability to have casual contact are problems that the DISEL team is trying to address.

3.3 Team Integration

At the start of the project, the project managers tried to plan activities to allow all of the team members to become acquainted with each other. The two project managers

took the first step themselves by having daily chat sessions. Through these chat sessions, they were able to become familiar with each others' management styles and philosophies. A close working relationship was formed soon after. Daily chat sessions and emails allowed the two team members to be in sync with each other. Very early on, it was decided that honesty between the project managers about the feelings and thoughts of the group was imperative if the project was to succeed. Many of the team interaction problems described in this document were revealed during these meetings between project managers.

The model under which the project managers operated to produce deliverables was for an idea or a document created by one person to be sent to the other for approval/comments. After agreement, it was put forth to the entire DISEL team. This usually resulted in a turn-around time of 48 hrs at the most. For more urgent matters, a talk session was held so decisions could be made instantly.

The first activity the project managers planned was a meeting in which all the team members would introduce themselves and tell everyone a bit about themselves. This did not work very well because there were communication problems and because there were no clear images of team member's faces to associate with names. There were no distinguishing stories or traits offered by any of the members, so no value was added by this exercise. The next activity for team integration was to require an email exchange between the team members within for each role, as well as a talk session. These did not work well either. Once the compulsion was over, the communications stopped as well. It seems that the members of a role team did not really come together until they had a deliverable to produce. The project managers established a good relationship immediately because they had immediate work to do. Similarly, the analysts, V&V team, and the QC team communicated frequently from the onset of the project because they had immediate responsibilities. Assigning a task to a team is the surest way to encourage the team members to communicate and forge relationships.

A more challenging problem is to get the team members from different roles to interact with each other. Throughout the DISEL project, there wasn't much collab-

oration between any of the roles until the implementation phase, when the designers and the programmers had to work closely together. Even at the end of the project, team members were more familiar with their fellow role team members than with other teammates. This appears not to be a big problem as long as there is communication between the roles when it is necessary. For example, at Microsoft, developers for an application will be much more familiar with other developers than with program managers or testers [8], but the groups still manage to collaborate well.

3.4 The Team Contract

A few weeks into the project, it became apparent that there was no standard expectation for many of the project's daily processes. Meetings scheduled through emails were missed because there was not enough advanced notice. Emails requesting information or action were not acted upon for days. Team members would be absent from meetings without warning. All of these mishaps occurred frequently because there was no set of guidelines on how these types of things should be handled. There was no formal agreement on who should make key decisions, whether the entire team needs to agree, or whether only a majority does.

In an attempt to formally document agreements on these and other issues, a team contract was drafted (Appendix A). This document established certain rules on communication, response-time, and authority [1]. For example, it was decided that all emails should be responded to within 24 hours, and that the project manager should be notified of any absences from meetings at least 24 hours prior. Internet talk sessions must be scheduled or canceled at least 24 hours ahead of time. For decisions having project-wide impact, the entire team must agree, but for smaller scale decisions, only the project managers and head of role teams need to agree. See Appendix A for the full text of the team contract. This document was put together and agreed upon by everyone in the team, both verbally, and via email. It is important that the project manager get written confirmation of agreement because a decision placed in writing is generally taken more seriously, and the project manager has

something tangible to refer to if there is any future conflict. This document should be put together as soon as possible to avoid some of the misunderstandings which occurred in this project. The team contract should not be seen as a rigid, frozen, document, but something that can be altered and broadened as new situations arise, similar to the US Constitution. A major mistake the project managers made in this project was failing to include consequences in case someone violates a term of the contract. This again, stems from a lack of real authority in the project manager position. It would be highly unpopular with the team if a student project manager seriously reprimanded a fellow team member. Under the current team contract, there were no consequences for violation the team contract, and no system for maintaining adherence. Basically, a team member would follow the team contract only if he or she felt like it. Granted, in a professional environment, this may be enough, but with absolutely no hint of consequences, a more cavalier attitude was taken by students. Not all emails were replied to in 24 hours. Meetings were still missed, though the frequency of these events lessened noticeably after the team contract was established. The project manager must also be diligent in reminding the team members of the terms of the contract. It may not be a bad idea to write the more salient points on a large poster and mount it on a wall in plain view by all the members of the team.

3.5 Project Schedule

3.5.1 Schedule Planning

Planning the project schedule was mainly an exercise in deciding how to allocate the nine month time period of the class between each of the phases of the development process. The waterfall method was selected as the process to use because of its simplicity and popularity. Cost estimation and scope estimation was difficult to calculate because the problem to be solved was so ill-defined. The task of developing a tool to facilitate distributed software engineering while immersed in such an environment forces the requirements to be almost constantly changing because as the project pro-

gresses, the team will recognize new, unforeseen issues and therefore, new, unforeseen additions to the requirements. Many times throughout the project, the project manager had to triage new ideas for the system and limit the scope of the project by putting off a feature for a future version. This vagueness of the requirements made the COCOMO method difficult to use, since code size was nearly impossible to determine. Instead of using this model, the guidelines of Brooks were followed and greater emphasis was placed on good design. More than 1/3 of the schedule was devoted to requirements, 1/3 was devoted to design, and slightly less thank 1/3 of the time was devoted to implementation/testing. A one week buffer was built into the schedule at the end for unforeseen circumstances (See Table 3.1 for complete schedule).

The large amount of time allocated to requirements gathering was to allow for a period of time to allow the team to become acquainted. There was assumed to be an initial period where productivity would be low as the team members got to know each other. The requirements phase was the only phase which had this introductory period built-in. In reality, this period was necessary in all of the phases because the team members did not get to know each other very well until they had to produce a solid deliverable.

3.5.2 Reallocating Resources Due to Personnel Changes

Through the course of the project, there were many personnel changes, both anticipated and unanticipated. The final team structure had significant changes from the original plan (See Figure 3-1). At the start of the project, the programmer role had no members from the MIT side. This was because there were not enough students at MIT to fill all of the positions. The plan was to draw programming talent from the MIT members holding other roles. Thus, at least two MIT members were anticipated to have dual roles during the implementation phase.

As the project progressed, several other modifications to resource allocation and scheduling were necessary. First, one of the validation and verification engineers resigned from the project. This did not significantly affect the schedule, but the resource allocations were shifted. One of the quality control engineers moved to

Task	Start	End	Team Responsible
Analysis Phase	9/11/97	12/9/97	
Create role work plan	9/11/97	12/2/97	Entire Team
Prepare to interview client	9/11/97	9/18/97	Analyst
Interview client	9/18/97	9/18/97	Analyst
Create requirements analysis document	9/19/97	12/9/97	Analyst
Present first draft of requirements	10/9/97	10/9/97	Analyst
Audit first draft of requirements	10/16/97	10/16/97	QC, V&V
Present second draft of requirements	10/23/97	10/23/97	Analyst
Audit second draft of requirements	10/30/97	10/30/97	QC, V&V
Present first draft of specifications	11/13/97	11/13/97	Analyst
Audit first draft of specifications	11/20/97	11/20/97	QC, V&V
Present second draft of specifications	11/27/97	11/27/97	Analyst
Audit second draft of specifications	12/2/97	12/2/97	QC, V&V
Present work plans for each role team	12/2/97	12/2/97	Entire Team
Freeze analysis document	12/9/97	12/9/97	Entire Team
Design Phase	12/9/97	2/12/98	
Present tests to be applied to design	1/9/98	1/9/98	Test
Present first draft of design document	1/15/98	1/15/98	Design
Apply of tests to design	1/9/98	1/15/98	Test, Design
Present results of tests	1/22/98	1/22/98	Test
Present second draft of design document	1/29/98	1/29/98	Design
Audit design document	2/5/98	2/5/98	QC, V&V
Freeze design document	2/12/98	2/12/98	Entire Team
Implementation Phase	2/12/98	4/9/98	
Code review	2/26/98	2/26/98	Programmer
Present test scripts and scenarios	3/5/98	3/5/98	Test
Code review	3/12/98	3/12/98	Programmer
Code review	3/19/98	3/19/98	Programmer
Final code review, Implementation complete	4/2/98	4/2/98	Programmer
Presentation of all test results	4/9/98	4/9/98	Test
System Integration	4/9/98	5/7/98	
Audit of entire system	4/9/98	4/16/98	QC, V&V
Present user manual	4/30/98	4/30/98	Documentation
System delivery	5/7/98	5/7/98	Entire Team

Table 3.1: DISEL Project Schedule

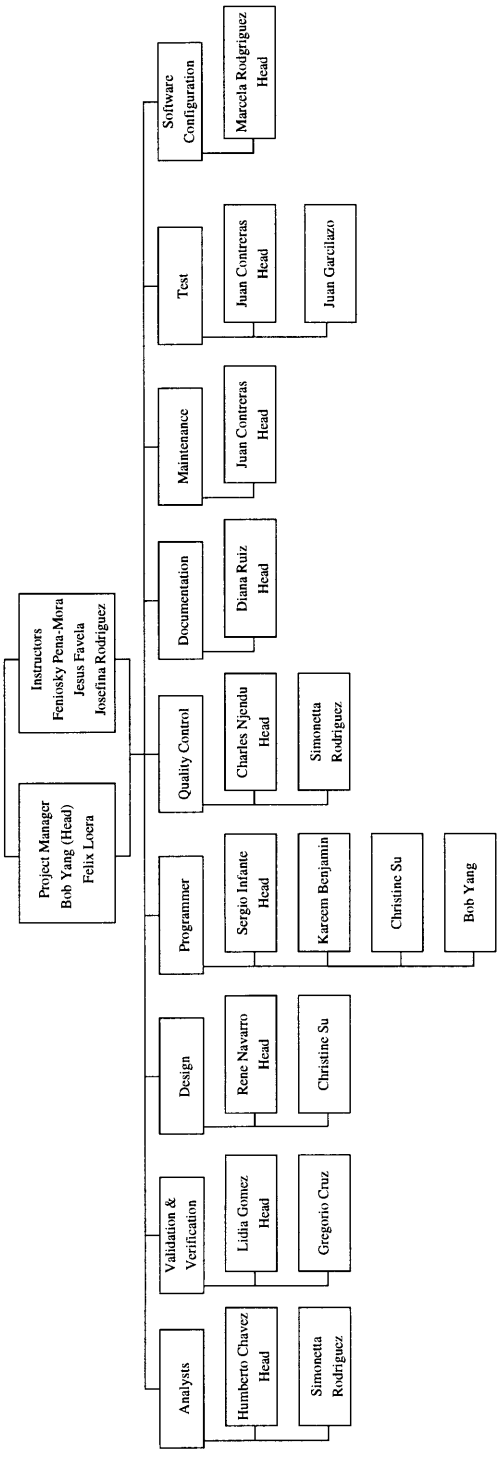

Figure 3-1: DISEL Organizational Chart at Project End

assist the remaining validation and verification engineer. This addressed the problem adequately and no large repercussions were felt.

Three months into the project, however, a larger resource problem was encountered. One of the design team members left the project. The project was still in the requirements analysis stage, but it was clear that the design phase would need more time allocated now that there were fewer people. The solution was to try to institute concurrency between analysis and design. As certain portions of the analysis were completed, they would be frozen and design work would begin on those frozen portions. In theory, this sounds very plausible, but in practice, design floundered until the requirements document was complete. Details of this period of work is discussed later in the summary of the design phase.

As the project entered the design phase, the head of the quality control team became ill and withdrew from the project. This was a serious blow to the project because there were only two quality control engineers, one of whom was helping with validation and verification tasks. Fortunately, the analysis phase had just been completed, so one of the analysts moved into the quality control team and assumed the role of quality control engineer. This worked out very nicely. The existing quality control engineer was able to bring the analyst up to speed in a short period of time and the QC team worked well together for the duration of the project.

During the implementation phase, one of the project managers and one of the designers from MIT were assigned part-time to the programmer role, and one tester was moved from test to programmer full-time. This was in anticipation of having a large task to accomplish in a short period of time.

3.6 Project Execution

3.6.1 Requirements Analysis Phase

The requirements analysis phase started off with the team attempting to become familiar with each other. Expectations and morale were both very high. The team

tried to start off by having video conferencing sessions where everyone introduced themselves, but these sessions did not work as well as anticipated due to language problems and the unfamiliarity of the team with the distributed meeting environment.

Team members within a particular role exchanged emails frequently in the beginning, but these emails tapered off between team members not actively involved in producing immediate deliverables (i.e. programmers, designers, and testers). It was also very difficult to have brainstorming sessions with both MIT and CICESE together. There was the feeling at MIT that it was difficult to explain many of the ideas to their CICESE counterparts because of the language barrier, and vice versa. Demonstrations and diagrams were not available between the two sides. In the future, electronic whiteboards, not computer-based, but actual whiteboards linked electronically would be ideal in facilitating brainstorming sessions.

Each role team was assigned the task of creating a workplan for the tasks required of their role. These work plans were an attempt to get the members of each role collaborating on producing a document. This initiative, however, faded into the background as the teams forgot about the assignment and the project managers never enforced its production. No team produced a work plan with concrete deliverables and estimated delivery dates. In the future, the project manager needs to be more proactive with reminding team of assignments.

Eventually, what happened was that the requirements process dragged on past the scheduled date and the team became bogged down with inertia. Successive iterations of the requirements document did not receive any constructive feedback from the team. The extent of comments made were for spelling and grammatical errors. The team left the entire burden of creating the requirements document upon the analysts. Morale slumped as more and more of the team felt that the project was not going anywhere, but the team members were unwilling to contribute ideas for the document. There was also a period of three weeks in December during which the two universities had winter vacation. No progress was made at all during this period and the team had trouble resuming work afterwards. Meetings and deadlines were missed, emails were not responded to. It took a full week for the team to regain focus and move

forward.

The two analysts had personality conflicts and stopped communicating with each other a few weeks before the winter break. The project managers were unsuccessful in reconciling the differences between the two analysts. There were language barrier issues, as well as personality issues. There was insufficient infrastructure in the project team to make the airing out of these grievances possible. Getting the the two members together to discuss things was not productive because the CICESE team member felt that he could not effectively communicate in English. One of the project managers felt that in the interests of maintaining the schedule, the analysts could work on different portions of the requirements document separately, so the analysis document was divided into two parts, with each analyst assigned to one part and no collaboration between them. Finally, three weeks after the scheduled completion date, the requirements and specification documents were completed. The final deliverable contained excellent, detailed content, but lacked any form of conceptual integrity. The specifications section describing the behavior of the program did not sufficiently explain how the requirements outlined in the requirements section would be satisfied. This was not unexpected considering the process through which these documents were created.

3.6.2 Design Phase

Headed into the design phase, the project managers realized that the original project schedule needed to be completely overhauled. In addition to containing inaccurate estimations of schedule, it did not contain enough specific detail for all of the tasks required of the team members. Consequently, a new project schedule was created (Table 3.2) complete with a Gantt Diagram (Figure 3-2) and a diagram of responsibilities (Figure 3-3). Though this schedule was not 100% accurate, it came much closer to actual time requirements and was much better at providing the entire team with guidance on what was required of them and when it was required.

The design phase was intended to operate in parallel with the requirements phase after one of the designers left the project. In practice, however, no useful progress

Role/Activity	Start	End
Design Team		
Design Social Interaction, UI	2/24/98	3/10/98
Design Casual Contact, Network	2/24/98	3/6/98
Programmer Team		
Establish User Interface (with Designers)	2/24/98	3/3/98
Implement User Interface (UI) and Social Interaction (SI)	2/24/98	3/17/98
Present first version of SI & UI	3/10/98	3/10/98
Implement Casual Contact, Network	2/24/98	3/10/98
Integrate all modules	3/16/98	3/16/98
Fix Bugs	3/23/98	4/5/98
Documentation Specialist		
Take meeting minutes	2/24/98	4/30/98
Maintain web page	2/24/98	4/30/98
Write user manual	3/26/98	4/16/98
Present progress	4/9/98	4/9/98
Test Engineer		
Finish bug tracking system	2/24/98	3/3/98
Design test cases for UI and SI	2/24/98	3/14/98
Implement test cases	3/4/98	3/20/98
Apply test cases	3/15/98	3/27/98
Design test cases for Network and Casual Contact	3/3/98	3/10/98
Implement test cases	3/10/98	3/29/98
Apply test cases	3/16/98	3/23/98
Validation and Verification Engineers		
Set agenda for each lab session	2/24/98	4/30/98
Perform validation testing	2/24/98	3/10/98
Prepare acceptance tests	2/24/98	3/10/98
Apply acceptance tests	TBD	TBD

Table 3.2: Revised DISEL Project Schedule

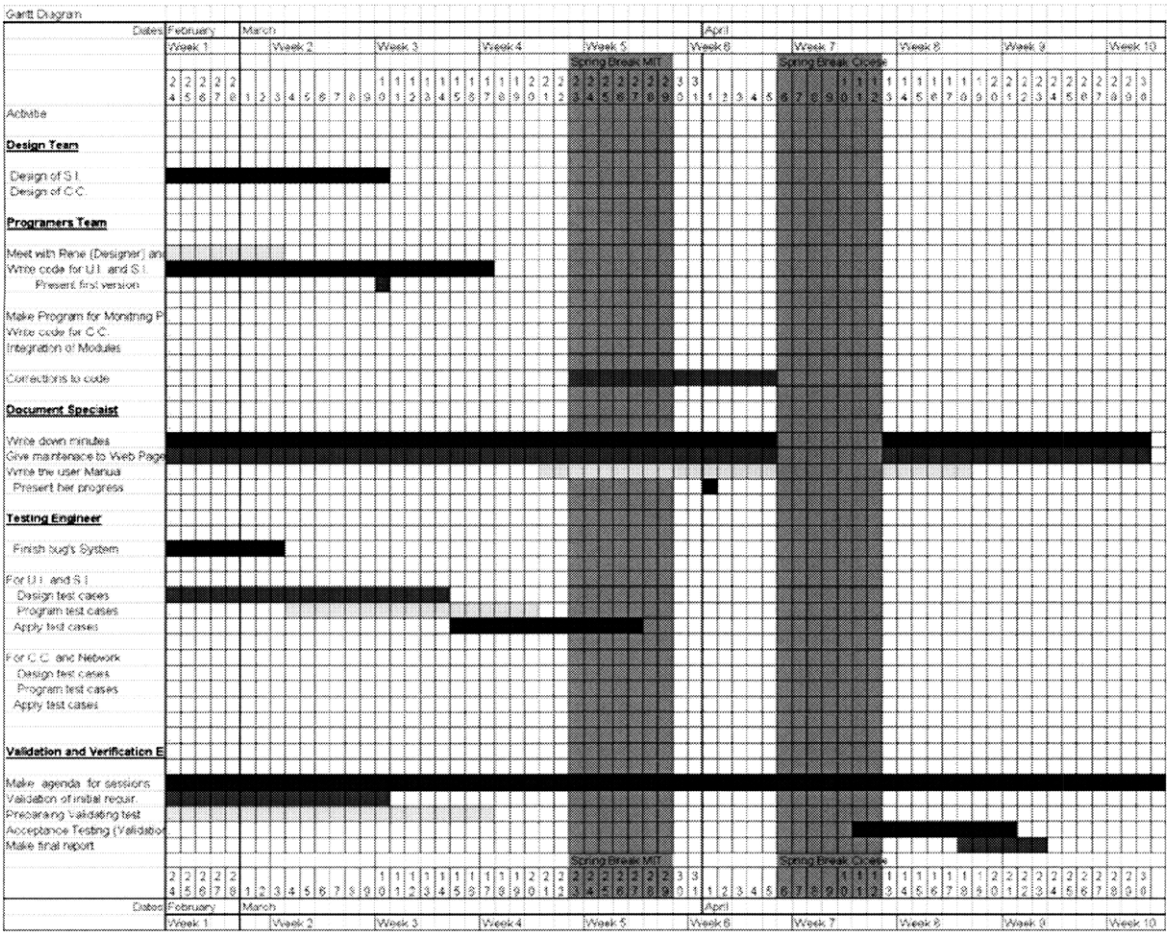

Figure 3-2: Gantt Diagram of Revised DISEL Schedule

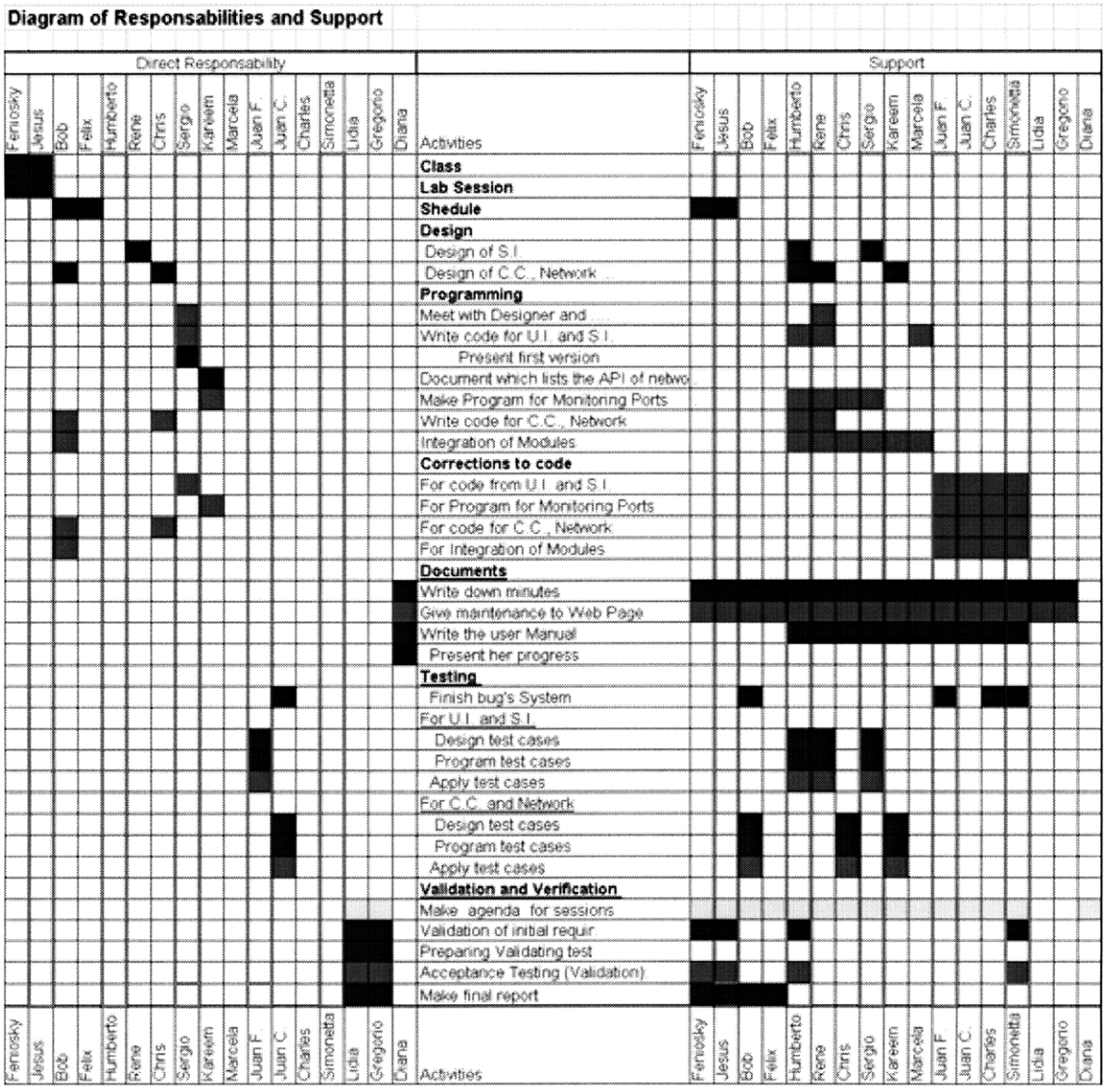

Figure 3-3: Responsibility Chart For Revised DISEL Schedule

was made until after the requirements phase was completed. There were two main reasons for this. First, there was still the sense of being in the requirements phase, so all of the attention was focused on the next iteration of the requirements document instead of on generating some form of design. The designers were not compelled to produce anything at all. Second, there was no overall vision for what the product should look like at this stage of the project. Ideas floated around, but they were not sufficiently fleshed out. The designers were to wait until a certain portion of the analysis document was completed before beginning design for that portion, but no portion of the document was frozen until the very end of the requirements analysis phase.

The design phase was very challenging and difficult because the design team was expected to inherit a vision of what the product should look like from the analysis document, but that document did not provide such a vision. The problem of envisioning what the product should look like landed squarely on the laps of the design team. A two-person design team should not be required to architect the conceptual vision of the entire application, as the designers were required to do in this case, without buy-in or suggestions from the rest of the group. The design team could not get any input from other team members at all because no one else felt responsibility. This is taking Brook's idea of one master architect to an extreme. There should be someone responsible for overall conceptual integrity, but there must be input, support, and buy-in from each member of the team, otherwise, there is little incentive for the rest of the team to contribute their effort, especially since there was no evaluation/reward system.

Due to winter vacations at MIT and CICESE, work on the design phase did not begin until the second week of January. With a hard completion deadline of April 30th, the schedule became very tight. The target date for completion of the design was set for the first week of March, leaving six weeks for implementation and test. It was a difficult task to balance the remaining time so that the programmers and testers had enough time to create a robust product while at the same time giving the design team enough time to create well-defined interfaces and modules so that the

programmers did not operate in total chaos.

The progress of the design team seemed to go well for the first three weeks, as both designers felt that the deadline was achievable. Successive iterations of UML (Unified Modeling Language) diagrams were created. Then, in the first week of February, the designer at CICESE revealed that he felt proper process was not being followed and that the design process should start over, this time using scenario diagrams and creating a feature list before actually creating an object model. It seemed like the team was afraid to make progress!

Several lessons can be learned from this incident. In this case, the project managers fell into the same trap as the rest of the team and assumed that since the designers said everything was okay and they were comfortable with progress, that this was in fact true. In scheduling milestones, there were only dates for drafts of the design document, there were no concretely defined progress attributes for the content of each draft. This made it difficult for everyone, the designers included, to gauge the progress of the design phase. As Brooks mentioned [6], it is critical that deadlines are set with well defined deliverables, instead of vague descriptions such as "first draft."

The second lesson to be learned is that the project managers must probe deeper and more frequently into the progress made by team members. Here, the element of casual contact was sorely missed. A simple, "Hi, how are things going?" asked by the water cooler is much more likely to elicit a frank response than a question at a formal meeting, especially given the aversion to creating unrest built into the CICESE culture. In a team like this, a project manager who is well-versed technically comes in handy. He or she should sit down with the designers and review progress and compare the evolution of the design specification with the requirements document.

Given the discomfort with the design and the tight schedule, something needed to be changed in order for any sort of final product to be produced. The project managers had to step in and try to fix a project that was spinning out of control. The team felt a sense of urgency, but was paralyzed to take action on this urgency. Each of the team members had become so ensnared by the boundaries of their role definitions, that they would not take any initiative to move the project along. Clearly, drastic

action was needed. The action taken was to divide the team into two halves, the MIT team and the CICESE team. The MIT was responsible for some aspects of the program's functionality, the network and casual contact portions, and the CICESE team was responsible for other aspects of functionality, namely the user interface and the social interaction portion. There would be no collaboration until it was time to integrated these two halves. Essentially, collaboration took second place to achieving product delivery.

This approach effectively created a rift between the two teams. Neither team had any idea of what the other was doing and any form of communication between the two sides outside of scheduled labs trickled to a halt. The designers proceeded with the design of their respective sections without defining interfaces between the two sections of the program.

3.6.3 Programming Phase

Once the teams had separated, the attitude at MIT was to begin implementing something as soon as possible. One of the designers took on programming tasks and the MIT project manager assumed the task of integrating the MIT and CICESE modules once they were mature and stable enough. Since there was no design integration whatsoever, the team operated on blind assumption of what the CICESE team would produce. The attitude was to worry about integration later, and to produce the functionality needed first, and then provide a level of abstraction through which this functionality can be accessed. The designer at MIT actually undertook a programming role, so the team fell into the unfortunate model of having design follow implementation. The design document was periodically changed to reflect the latest code generated instead of the other way around! This is clearly very poor methodology, but the QC did not sufficiently take issue with this defect in process. Basically, what was happening in March was that the team waited for the programmers to produce something that they could review and test. There was also a one week period in the last week of March which was MIT's spring break. Then two weeks later, CICESE had their spring break. These breaks were not as devastating as the winter break to

productivity because of the separated nature of the development process. While MIT was on vacation, CICESE continued to efficiently produce code, and vice versa.

The code came out fast and furious, and a phenomenal amount of work was done in a very short time. The CICESE and MIT teams produced outstanding work, but neither side had access to the other side's code. The CICESE side never saw any of the work that the MIT team did because they were unable to properly set up the MIT prototype. Another difficulty in operating in a distributed environment revealed itself here. Diagnosing software bugs without being able to have someone demonstrate the bug in front of you is an exercise in futility. It was almost impossible to determine what the problem was if CICESE got a NullPointerException error in Java. Owing to these problems, the CICESE team had absolutely no idea what the MIT team was doing at all. To make matters worse, both of the test engineers were at CICESE, so the MIT code was never thoroughly tested! The CICESE demo fared better at MIT and the team members were very excited by the progress when it was demonstrated. It was likely that CICESE would have been much more encouraged had they been able to see the MIT demonstration.

3.6.4 Integration

When the two teams felt that their modules were sufficiently mature in functionality and stability, one of the members of the MIT team set out to integrate the two pieces together. The tricky issue here was that none of the programmers knew what the programmers on the other team was doing. The MIT programmers did not know anything about the code that the CICESE programmers wrote and vice versa. Initially, only one person, the integrator, understood the structures and modules of both teams. When bugs or changes were requested, the fixes often created more problems than they solved because the programmer creating the fix was ignorant of the effects of the changes on the system as a whole. A poor design document due to a rushed design phase was the largest contributor to this problem. The balance in scheduling that the project managers had tried to achieve erred in allocating too little time for the design document to fully mature.

Integration was also difficult because there was no common code repository where everyone could download the latest versions of the code. There was also no infrastructure to manage version control of code. When the integrator requested a change or a bug fix, he often received new versions which contained code based on modules as they existed before the integration began. For example, the CICESE programmer did not download a new version of the integrated code until the last week of the project. The integrator had to reconcile differences between the patched code he received and the existing integrated code he was working on. In the future, all programmers need to be aware of the other modules and interfaces contained in the product. He or she does not need to understand the code line-by-line, but he or she does need to know at a high-level how the code operates.

The last week of the integration phase was the period of time in which everything fell into place. The team operated better during this time than any other period of the project. There was a new version of the integrated code available each day. An installation program was created allowing everyone to finally see the system functioning. The team was operating from code stored in one central location, with bug fixes appearing within 24 hours. The project managers enforced a rule that any code changes made must be made on the latest build available that day, or the code will not be incorporated.

The only weakness was the fact that no formal bug reporting system was used. A system was created to report and track bugs, but the project never reached a point where formal testing occurred. The test engineers did not successfully install the complete program until the last week of the project, so all of the testing was done by the programmers on their own code.

3.6.5 The Final Product

Once the deadline date was reached for the completion of implementation, the validation and verification engineers did a formal comparison of the final product against the requirements document and discovered several requirements unsatisfied. The project had proceeded in a manner which caused the team to lose touch of the initial

requirements. Analysis and design consumed so much time that in the end, there was only a rush to produce something that worked and satisfied some of the requirements. Narrowing project scope in order to meet an unalterable time constraint resulted in leaving many of the requirements to be addressed for a future version of the project. In the end, the product was successful in providing a way for users to engage in casual contact in cyberspace, but the social expression aspect could have be improved. The system was also not very stable because it had not undergone enough rigorous testing.

3.7 Lessons Learned

Looking back at how the DISEL project progressed, many good things happened which enabled the team to produce a final product that delivered most of what it set out to deliver, namely to enable geographically distributed users of the system to engage in casual and social interaction in a manner which simulates face-to-face contact. There were many useful lessons learned which, when applied to future endeavors, can prevent some of the stumbling blocks and slowdowns encountered in this project.

One of the key reasons why the team had so much trouble recovering from the schedule slippage during the analysis phase was that the team members were tightly compartmentalized into their role. They owned a part of the software engineering process instead of the product itself. The result of this was that no one felt obliged to assist the analysts. In speaking with the team members afterwards, all of the roles but the analysts felt that they did not contribute at all to the requirements and specification documents and felt no ownership of the project until the project required a deliverable from their role. This means that no one in the team other than the analysts felt that they had contributed anything to the document which established the vision and direction of the entire project! In the future, the roles and responsibilities of each team member cannot be so narrowly defined. Each member should own the product first, and the process second.

Another reason for the schedule slippage was the lack of communication between the team members. As mentioned previously, the element of casual contact was sorely missed. There was no opportunity for the team members to informally toss around ideas and reveal worries and problems to each other. Synchronous communications (real-time talk and response) occurred either in a formal lab setting in front of the entire team and instructors, or in chat sessions which had to be pre-arranged at least 24 hours in advance. In order for a team to feel close and communicate effectively, daily synchronous communication must occur. Email is not enough. The project manager should establish certain times of the day when team members are guaranteed

to be available to talk. Currently, a team member has to schedule a talk session or send an email every time an idea or question pops up. He or she will likely just leave the issue open and unaddressed since it is so much trouble to communicate with counterparts, and the communication delay is so long. The solution to the problem is to establish work periods of 1-2 hours when team members come to class to work on the project. These periods will not have an agenda, and the team members do not even have to work on project related material, but they must make themselves available for communication. These sessions should at occur at least twice per week in addition to the formal labs. In the DISEL project, it was often the case that 24-hours were wasted because one of the team members was waiting for his/her counterpart to review a work product.

Finally, for the project managers, lessons can be learned in the art of project scheduling. In this project, the design phase was definitely too short to produce a good design with well-defined modules and interfaces. The rush to begin implementation resulted in headaches down the road when the separate modules had to be integrated. Future schedules should not allow requirements analysis to spiral out of control for so long. At some point, the project manager needs to step in and say "we are proceeding with what we have right now," instead of allowing the process to drag on. This will eliminate the time pressure imposed on the design and implementation phases.

Chapter 4

Recommendations

Looking at all of the changes proposed throughout this document, if one were to undergo another project similar to DISEL, a very different approach would be taken.

4.1 Team Structure

Observing the behavior of the DISEL team, it appears that the traditional role based team structure is not the optimum way to assign responsibility to team members. The plan of attack outlined below is adapted from the Microsoft approach to software engineering as documented in [8, 14]. This approach focuses on getting every team member to have a personal stake in shipping a quality product on time. There will only be three roles to which a team member can be assigned to: Program Manager, Software Development Engineer, and Quality Engineer. There will be one additional role, that of vice president to be occupied by the instructor.

The organization of the project is dynamic throughout the evolution of the project (Figure 4-4). Each team member will be assigned to both a role team (Figure 4-1) and a feature team (Figure 4-2). A feature team is a group of people responsible for a driving a set of features from conceptualization through implementation and test. During some phases, team members will focus on the responsibilities of their roles applied to the entire system as a whole (a process oriented organization). During other phases, the team members will focus on the responsibilities of their roles applied only

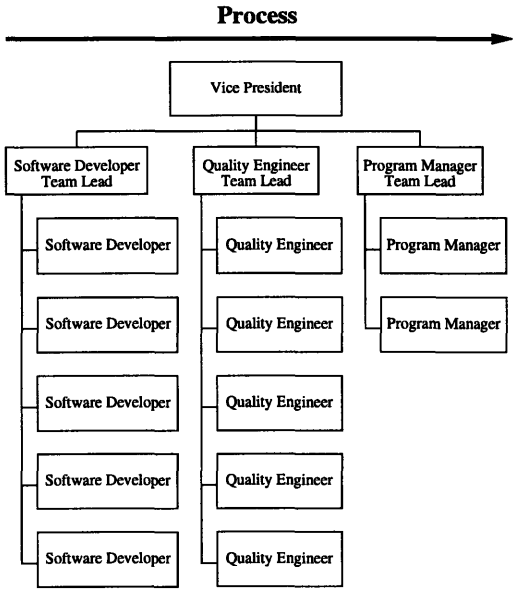

Figure 4-1: Team Organization: Role Based Teams

to the set of features which falls under the umbrella of their feature team (a product oriented organization) (Figure 4-3). Though the scope of focus will change, a team member should not lose sight of the big picture while focusing on the feature set and vice versa.

4.1.1 Program Manager

The program manager is primarily responsible for formally creating the requirements, specifications and feature descriptions of the final product. This involves being in touch with the marketplace and potential users. The program managers will be the ones interviewing the client to ascertain requirements. The program manager is also responsible for coordinating between different groups in order to resolve design and feature conflicts. Finally, the program manager should make sure that all of the little details, such as creating a setup program, the user manual, and a bug database are taken care of. Ultimately, the program manager is responsible for creating a vision of what the program should look like since he or she is the one responsible for the high-level specifications and feature descriptions.

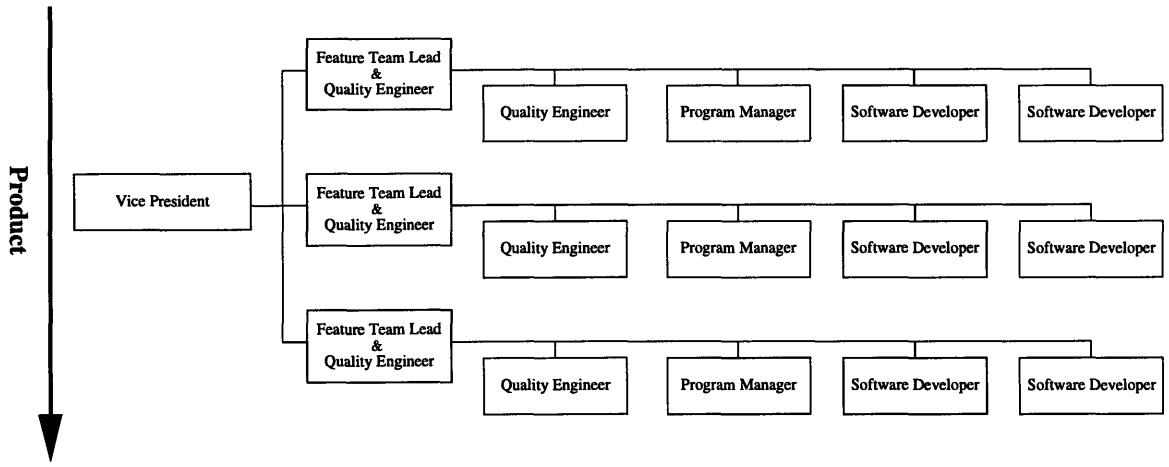

Figure 4-2: Team Organization: Feature Based Teams with Quality Engineer Lead

The ideal candidate will have good communication skills and be good with administrative tasks. He or she must have a knack for understanding what features are useful and how to make them easy to use. Finally, the successful program manager must have a basic technical understanding of technology and code in order to effectively match the scope of the specifications with capabilities of the programmers. Unlike the traditional role of project manager, it is not the sole responsibility of program manager to create and enforce the project schedule. This is something that must be negotiated and monitored jointly with the developers and the quality engineers. In no way is the program manager boss over the other two roles.

In a team of 15 people, there should be three program managers, with one serving as the lead program manager. The program manager lead is additionally responsible for making sure that the conceptual integrity of the program is maintained, as well as working with the lead developer and lead quality engineer to create an overall schedule and workplan. For large decisions impacting the entire project, the three leads will get together and represent the points of view of their respective role teams to make a decision.

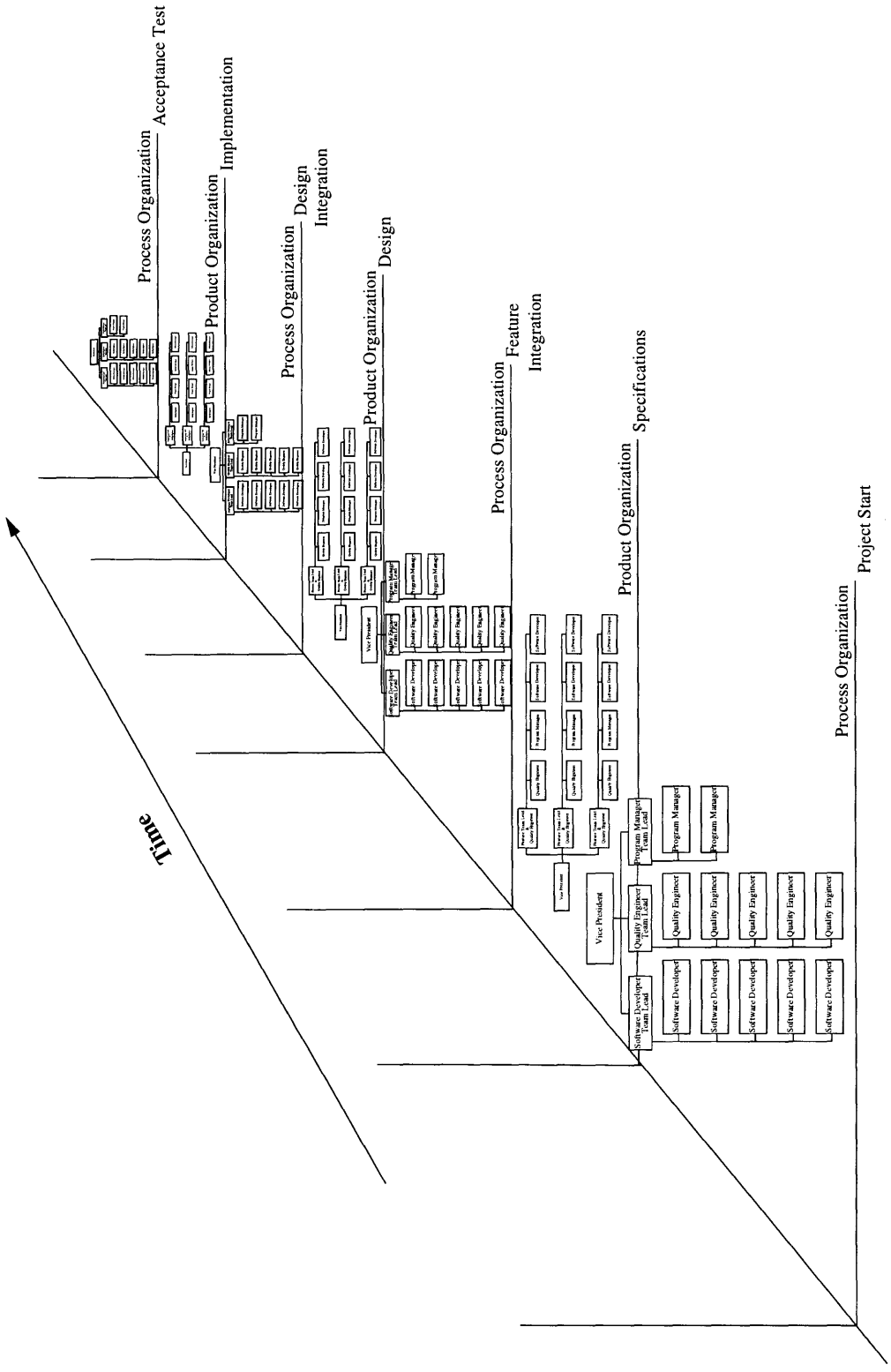

Figure 4-3: The Dynamic Nature of the Team Organization

4.1.2 Software Development Engineer

The software development engineer (developer) is responsible for the low-level modular design and implementation of the project. During the initial definition of specifications and features, the developers must give their assessments of how difficult a particular feature will be to implement and whether the technology is available or must be developed in-house. The developer must then translate the high-level feature list generated by the program managers into a low-level, object-by-object design, and then implement this design.

The idea candidate will have excellent design and programming skills. He or she will be familiar with UML, Java, and C++. The development engineer will take on the traditional roles of both designer and programmer, so knowledge of good object-oriented methodology is very important. Managing the scope of the project is an important duty that the developer must assume. If the developer thinks a feature is difficult to implement, or a bad idea, it is his or her job to tell the program manager.

In a team of 15 people, there should be six developers, with one serving as the lead developer. The lead developer has the additional duties of creating the development schedule and making sure that the design of each module has well-defined interfaces so that integration will not be a disaster. The lead developer should also be the most technically competent member of the development team so that he or she can answer questions from other developers and assist other developers, if necessary, to meet a schedule deadline. The entire development team is responsible for creating a central code repository where version control can be enforced. Synchronizing new code and running daily builds is also the responsibility of the development team.

4.1.3 Quality Engineer

The quality engineer is responsible for verifying the quality of everything produced throughout the project. They are also responsible for maintaining the documentation repository and ensuring that all decisions and ideas are documented properly. Finally, they are responsible for setting the agendas of team meetings. During requirements

gathering and feature design, the quality team must make sure that requirements are reasonable and in touch with the market. They must make sure the product is neither overly ambitious nor underambitious. During initial scheduling, they must make sure that milestones are well defined and detailed. During the design phase, they must make sure that all of the modules can be integrated and that there is no functionality missing or duplicated. During the implementation phase, the quality team must test the code for bugs and make sure that the initial stated requirements are being met. Test scenarios and performance criteria should be defined early on in the implementation phase. Finally, the ultimate responsibility for meeting the delivery dates rests on the quality team. If they feel that a particular phase of the project is stagnating, or that the project is no longer on track to be of high quality, then they must alert the team to take action.

The ideal quality engineer will be a meticulous person, able to sift through small details that others might miss. This person must also be able to approach problems from a wide variety of viewpoints in order to find problems which are not obvious. A quality engineer must also be an outspoken person because often times they must be the ones to infuse the reality of deadlines and schedules into a process that is out of control.

In a team of 15 people, there should be six quality engineers, with one lead quality engineer. The lead quality engineer will be responsible for setting the direction of the quality control process and be constantly in touch with the progress of the project. At any given moment, the lead quality engineer should know how far each module of the project is progressing and where the weaknesses are in project progress. The quality engineer role takes on many of the functions of a project manager in terms of keeping the team on schedule. A strong lead quality engineer, one who will maintain proper scope and prevent feature creep, will be very useful in ensuring that the project is completed on-time and of high quality.

4.1.4 Vice President

The duty of the vice president is to monitor the progress of the project. The role should be occupied by the instructor. He or she should be in weekly communication with the three team heads to discuss activities planned for the week and progress from the last week. The job of the vice-president is not to make decisions for the team, but to make sure that the team is on track and not headed towards disaster. The role of vice president is used to exert authority in case a team member is not performing up to expectations. In the beginning of the project, the vice president should review deliverables carefully to make sure deadlines are being met and the work is of high quality. This will establish a precedent of having the instructor review student's work, heading off the problem of students cutting corners if deliverables are only being reviewed by peers.

4.2 Project Execution

The first month of the project should be entirely devoted to instructional material. The team will become familiar with the theory of software engineering and practice working together through a series of collaborative problem sets. This helps the team members get to know each other before the formal start of the project. Suggestions for problem sets include creating a document repository for use later in the project and creating a bug tracking database to report and assign bugs.

Once this instructional period is over, the team members are asked their preferences for which of the three roles they would like to assume. The vice president takes these preferences into account when he or she assigns each person to a role. The vice president will also pick the heads for each role, based on the criteria defined previously.

4.2.1 Requirements Analysis Phase

Once the team organization has been decided, the project will formally begin with the team as more of a process-oriented organization, with each team member primarily focused on applying their skills to the entire system as a whole. The team is presented with the problem to solve, namely to develop a tool which improves communication between people in different geographical locations. The program managers are responsible for interviewing the client and performing formal requirements gathering.

Each team member will then have one week to think about the problem and submit at least two ideas on how to solve the problem. Each person will have a different interpretation of the problem and how it should be solved, so many different ideas will be presented. Once the ideas have been collected, a team-wide meeting will be held to evaluate and discuss each idea. A long block of time should be allocated for this task. Every idea will be discussed and the team will vote on whether to incorporate an idea into the product or not. At the end of this process, there should only be two or three main ideas accepted. The list of ideas should not be small, well-defined features, but broad functional definitions. For example, a program which provides the ability to convey a user's facial expressions to everyone else using the system is a good, broad idea. How many expressions to allow and how the user will change the expression is too detailed and should be merged with the first example. The goal of this exercise is to set a vision for the product.

Feature Groups

Once the main ideas for the program have been decided upon, all of the ideas will be divided into thirds. The project team will split into three subteams, called feature groups, adopting a more product-oriented team structure. There should be one process role lead in each feature group so that no feature group feels special for having more role leads. Each feature group is essentially a small-scale model of the entire project and will consist of one program manager, two developers, and two quality engineers. The quality engineers are in charge of making sure the feature team pro-

duces deliverables on time. Consequently, the feature group should select one quality engineer to function as the product group lead. Each feature group is responsible for 1/3 of ideas/features established during the requirements phase for the duration of the project. This entire process of requirements gathering should not require more than three weeks to accomplish.

The DISEL project consumed far too much time in the requirements analysis phase. The main ideas in the requirements document were present two weeks into the analysis phase. The rest of the time was spend elaborating on these ideas, adding little value in proportion to the time consumed. This approach sets the vision on the team very early in the project and immediately forces the team to begin realizing this vision, instead of waiting for the vision to be endlessly refined, causing momentum to be lost.

4.2.2 Specification Phase

Once the requirements have been voted on and defined, the entire team should decide how long to allocate for defining specifications and creating a list of features which will implement the main ideas. The purpose of this phase is to flesh out and refine the main ideas. This phase should not take more than four weeks. At the start of the phase, the team members will narrow the scope of their responsibilities to focus primarily on the tasks of their feature team.

Within each of the feature groups, all of the members should meet and propose features and attributes for their area of responsibility in the program. Then, they should rank the features according to importance. Ranking the features is important because it prioritizes the work in case the schedule begins to slip. In ranking the features, each team member should take into account the responsibilities of his or her role. The program managers should be most concerned with how useful this feature will be to the user. The developer should be most concerned with how long and how difficult it will be to implement this feature. The quality engineer should be most concerned with how each feature fits in with the rest of the program and checking the program manager's assessment of usefulness and developer's assessment of ease

of implementation.

At the end of this phase, the focus of each team member will shift back to the system as a whole and the entire team will review the list of all the features and add or remove items, based on how well they integrate with the rest of the system. A formal meeting should be held to perform this task. During the meeting, the lead program manager and lead quality engineer must make it clear that there should be a shift in focus to the system as a whole, as the team refocuses itself into a process-oriented hierarchy. The lead program manager and quality engineer will be in charge of integrating the features.

This method of setting features and specifications has the benefit of ensuring that all team members contribute to the process. Everyone is expected to contribute their ideas, giving them a sense of ownership in the program. It also prevents thrusting the entire burden of specification design on one small group of people, while the other team members sit back and wait.

4.2.3 Design Phase

Once the lists of features are ranked and finalized, a project schedule needs to be developed. The project schedule should be created by the role heads jointly, after consultation with their role team. The project managers focus on when the formal specifications will be completed and the programmers focus on when the module-level design will be completed and when implementation will be completed. The quality engineers will make sure that these schedules are realistic and that the milestones are well-defined. For example, it would not be realistic for programmers to begin implementation before the formal specifications have been completed by the program managers. The final deadline for delivery of the system must be set for the last week of April, due to the constraints of the academic calendar.

Once the schedule is set, the program managers and quality engineers narrow the primary focus of their work back to the responsibilities of their feature team. The team again shifts into the product-oriented organization. Program managers begin to formally document the exact behaviors for each of the features assigned to his/her

particular team. Scenario diagrams and justification for the features should be a part of the specification. Mock dialogs and screen shots should be incorporated where applicable. Collaboration between program managers is important to ensure that the program has a common look and feel. Specifications may change later, but the exercise of formally writing the specification forces the program manager to thoroughly think through an idea.

The developers remain focused on the system as a whole and must work closely together across feature groups to establish an object framework for the application. Once that has been completed and reviewed by the quality teams, they too focus on their feature groups again and proceed to design the objects and modules for the set of features they are individually responsible for. The first thing each developer should do is publish a set of interfaces for his/her modules. These must be frozen early in the design process. Then, as the design diagrams become more and more detailed, the development lead must review the interfaces for duplication and/or inconsistency. He or she, along with the quality engineer lead, are ultimately responsible for the integration of the object modules.

The quality engineers have a very important role during this phase. They must monitor the work produced by both the developers and the program managers. Specifications should be reviewed for ease of use, alignment with the rest of the system's features, and thoroughness. The diagrams and interfaces produced by the developers should be reviewed and compared to the specifications to ensure that they are in agreement. They should also be reviewed for adherence to good object-oriented principles. The quality engineers need to work closely together and be very outspoken with their opinions.

With six people working on module design and three people working on specification design, this process should not take more than six weeks. Due to the highly collaborative nature of this phase, daily communication is of the utmost importance. It is critical that team members make themselves available for consultation at predetermined times of the day.

The end deliverables of this phase are formal specifications and a detailed, module-

level design. Within each feature group, the quality engineers are responsible for making sure these deliverables are completed on schedule. The lead program manager and lead developer will manage the production of the deliverables and aid the quality engineers in schedule monitoring, as they can provide better insight on the progress of their role teams and what needs to be done if a problem arises.

Winter Vacation

Winter vacation will fall in the middle of the implementation phase. There will inevitably be a slowdown in productivity during this time. The best way to manage it is to have team members perform some thinking on their area of product responsibility. The break will occur after the critical specifications integration period, so most of the team members will be focused on their own feature set and can work fairly independently. One must be careful to resist the temptation to assign formal work because a vacation is probably more than due. Instead, encourage the team to consider issues of integration and potential problem spots. The developers will have to be careful here because they must work closer together than the other teams. The lead developer must especially consider integration progress and the challenges ahead.

4.2.4 Implementation

Once the quality engineers have decided that the specifications and design are comprehensive enough, the implementation phase can begin. Once this phase has been reached, all of the design and specifications should be so well integrated that any programmer can implement the design and be able to create the final product. If this is not the case, then more time needs to be spent on the design.

Before the implementation formally begins, the programmers should have created a central code repository where team members can get the latest version of the code. Facilities should also be provided for version control, so that two people cannot modify the same piece of code at the same time. Once the programming starts, each developer will have one quality engineer working closely with him/her and review produced code

daily. This way, implementation and testing are done simultaneously. There is no formal test phase following the implementation phase. During the beginning of this phase, the main focus of the developers and quality engineers is on generating code. Towards the end of the phase, the main focus will shift to testing, so that developers, quality engineers, and program managers are all testing the product.

Throughout the implementation process, the program managers are reviewing the progress and making sure that the implemented product is following the specifications created during the earlier phase. They should also be writing the user manual, since it is a natural evolution from the specifications document. If a change arises, the lead developer, program manager, and quality engineer act as the Software Configuration Managers. They will consult with the feature team under whose umbrella of responsibility the change falls under, and triage the change request. If the project is falling behind schedule, the configuration managers have a ranked list of features from which they can prioritize and cut features from. The ultimate responsibility of making sure each feature team delivers rests on the shoulders of the quality engineers. The lead quality engineer will have the additional responsibility of making sure the entire system is delivered on schedule.

Spring Vacation/Holy Week

During late March and early April, MIT and CICESE will have a break of one week. The teams break at different times, with MIT leaving the last week of March and CICESE leaving two weeks after that. The team assignments should be made so that for each functional role in a feature team, there is one member from MIT and one member from CICESE. The exception would be the program managers, of which there is only one per feature team. This allows for work to progress while one of the functional members is absent. The independent nature of the feature teams allows for very few interdependencies across feature teams, especially since everyone is working from one integrated design. The team members on vacation should reflect on the work remaining and try to identify potential hurdles and methods to overcome them.

4.2.5 Acceptance Testing

At the end of the implementation phase, a stable, robust product is expected. The final round of tests will focus on how well the program satisfies the original requirements as stated at the beginning of the project. The product should also be judged on conceptual integrity and ease of use. This round of tests should not take more than a few days as it consists primarily of using the produce from a user's standpoint.

4.2.6 Discussion of Project Plan

Goals

There are two primary goals that this project plan (Table 4.1) tries to attain. The first is to have every team member deeply involved in all aspects of the project. One of the weaknesses of the DISEL project was that most team members were active only for a small portion of the development cycle. Programmers and testers did not have very much work until the very end of the product cycle. In this plan, every team member has work to do during the entire development cycle.

The second goal of this project plan is to make sure each team member feels a strong sense of ownership and contribution towards the final product. Everyone is expected to contributed their ideas to the vision and feature list of the product. This team-wide empowerment gives an incentive for each member to stay interested in the product. The paradigm shifts from ownership of a small process in the development cycle to ownership of the product itself.

Scheduling

Although the team organization is radically changed from the DISEL model, the scheduling and the development process still loosely follows the waterfall method (Figure 4-4). The problems faced by the DISEL project did not stem from the development methodology, but from poor execution. In the proposed project plan, the requirements analysis phase is set at three weeks. Whatever the team can accom-

Task	Team(s) Responsible
Requirements Analysis Phase	3 weeks
Establish web document repository	Quality Engineer
Establish team contract	Entire Team
Brainstorm list of key big ideas	Entire Team
Focus vision of team on 2-3 ideas	Entire Team
Divide team into feature groups	Entire Team
Audit team divisions and ideas	Quality Engineer, Vice President
Specifications Phase	4 weeks
Propose features for system	Entire Team
Finalize and rank features to be incorporated	Entire Team
Audit final feature list	Quality Engineer, Vice President
Design Phase	8 weeks
Create formal project schedule	Role Leads
Design object module of system	Developers
Formally document specifications	Program Managers
Audit specifications and object module	Quality Engineer
Create interface definitions for modules	Developers
Audit interface definitions	Quality Engineer
Implementation Phase	10 weeks
Create bug tracking system	Quality Engineer
Create common code base	Developers
Implement modules	Developers
Test code	Entire Team
Bi-weekly code reviews	TBD
Create user manual	Program Manager
Conduct usability tests	Program Manager
Update specifications as necessary	Program Manager
Acceptance Testing	1 week
Final usability and code testing	Entire Team
Acceptance testing	Vice President, Quality Engineer

Table 4.1: Suggested New Project Plan With Approximate Time Durations

plish in three weeks will have to be enough. In DISEL, there was simply too much time devoted to rooting out every detail in the requirements. This detail can be extracted as needed during the specification phase. The difference between detailed analysis during requirements gathering and detailed analysis during specification is that the work done in the specification phase is a hard concrete step towards shaping the system, whereas work on the requirements definition document serves only as a guideline on how to attack the problem. When details are resolved in specifications, it translates directly into objects or methods, a concrete, tangible application of the resolved detail. Details resolved during requirements analysis do not have the same direct, tangible application.

The specification phase and the detailed design phase represent a separation of the original DISEL design phase. The specification phase should be very short and focused. Whatever features the team can propose in the span of four weeks should be more than adequate. Again, the plan seeks to avoid bogging down the entire team while small details are resolved. The detailed design phase is used to resolve the details of these high-level ideas. The difference is that three teams are working in parallel to flesh out these ideas instead of just two people handling everything. The tyranny that Brooks proposes is used here, only one or two people are responsible for all of the details related to a particular idea or feature. Each subteam of five people is responsible for an entire subset of the program. Others may make requests, but one team owns it and has the final say in shaping it.

The creation of a formal project schedule *after* the specifications phase allows for a more accurate estimation of the schedule. At this point, the program managers and developers understand the features they are trying to implement and have a better grasp of the work at hand. They simply need to divide the remaining time into a period for design and a period for the combination of implementation and test.

The implementation phase is grounded on the premise that implementation and test are intimately tied together and inseparable. It does not make sense to complete implementation before a formal test phase can begin. It makes more sense to have a developer/quality engineer pair closely working throughout the implementation phase

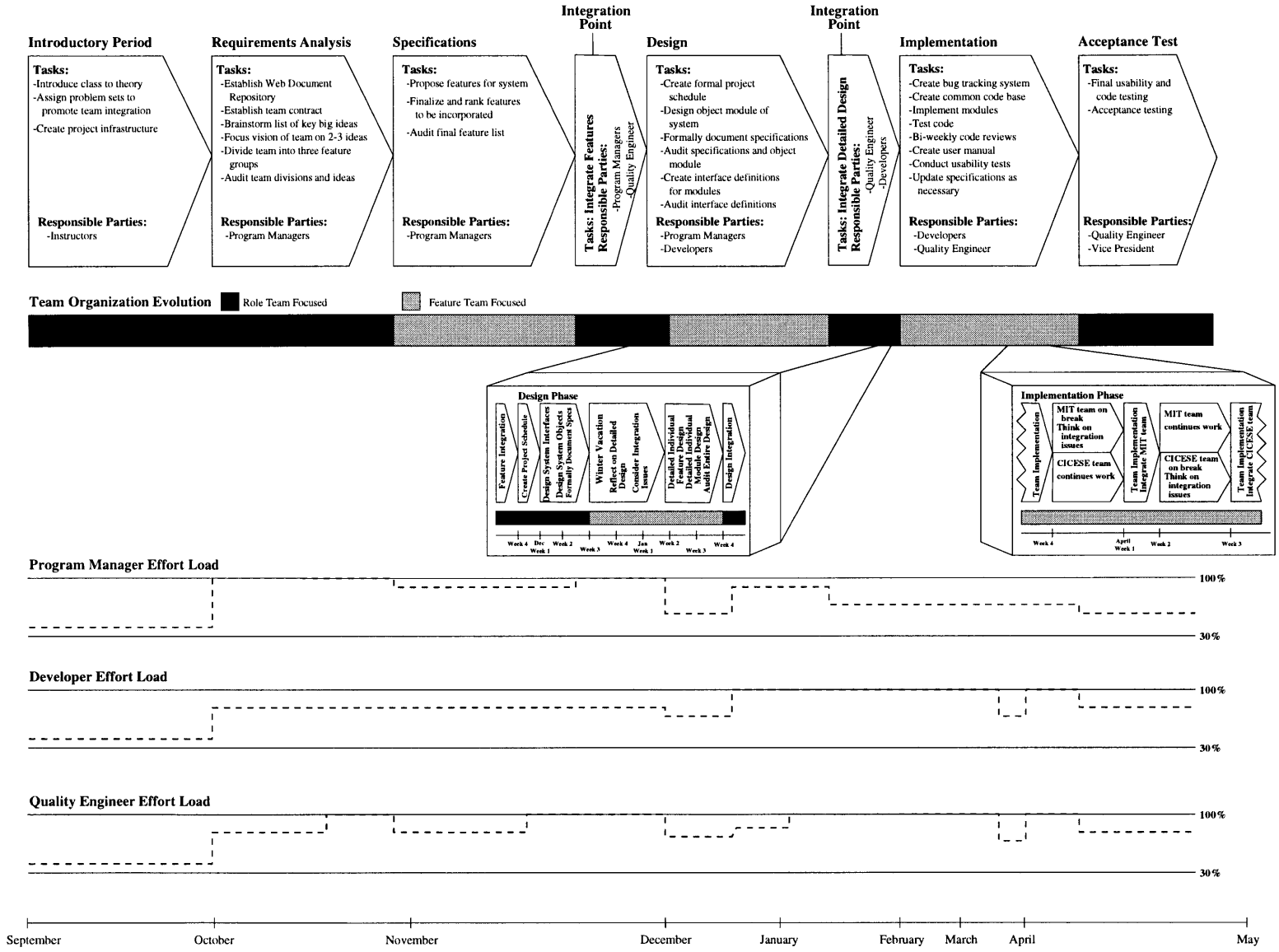

Figure 4-4: Phases of Proposed Project Plan

to deliver quality, defect-free code. As soon as it is feasible, a new build of the integrated program should be done daily and system-wide tests performed on this build by everyone on the team. Each member will focus most closely on the set of features his/her team is responsible for, but will also be able to test how well these features integrate with the rest of the system.

The acceptance test phase is a very short period of time. In this phase, the complete system is tested from the standpoint of a user. This is the validation and verification portion of the development cycle. Small user interface tweaks are still possible here to put the last bit of polish on the final product.

Potential Problems

One of the biggest anticipated problem areas with this plan is the difficulty of integration. Each subteam is responsible for a subset of the main program, and will drive it from beginning to end. Though there are different areas of primary responsibility, the group is still one team and constant communication is important. The goal of assigning responsibilities is to ensure that there is one person or team to whom everyone can go to with questions on an area of the program. There is no opportunity for fingerpointing and passing responsibility around. The drawback to this is that integrating the different areas of responsibility and functionality into one cohesive product will be very challenging. The main integration points will occur at the end of the specification phase and the detailed design phase. These are critical junctures in the project, when team members shift their focus from the narrow scope of their subset of features to the broad scope of the system as a whole. Strong leadership and attention to details by the role leaders and quality engineering team is vital.

Another potential problem area is with team dynamics and collaboration. This project plan budgets one month in the beginning for the team members to get to know each other while learning the theory behind software development. Then, they are thrown into the project with the first deliverable due in three weeks. The urgency of this deadline should bring the team together in close collaboration very early on. This can fall apart if there is not daily communication between the team members.

The no-agenda, casual lab sessions proposed earlier in this document are absolutely vital. Email is too slow of a medium to be used exclusively for communication during the requirements phase of the project. The type of urgency created by having a portion of the schedule be based on hard dates rather than attained progress is a necessary tradeoff to establish momentum early in the project. There is a danger of having the project completely lack direction due to poor requirements analysis, but that danger is slim if the DISEL project is any indication. The central vision of the DISEL product was established very early in the requirements analysis phase. The role heads and the vice president must strictly enforce constant communication, even if it is only to say "hi."

Chapter 5

Conclusion

The DISEL project set out to discover the pitfalls and difficulties associated with developing software in a geographically distributed environment. The team was modeled after a software engineering company and team members were assigned individual roles and were responsible for a particular aspect of the project. The goal of the team was to produce a software tool that improves the ability of people to communicate across long distances. Issues such as facial expressions, body language, and the opportunity for chance encounters were all investigated and most were incorporated into the final program.

The exercise of going through an entire development cycle with a distributed team uncovered many new and interesting issues that project managers must address. The most important of these are team integration, language integration, and cultural integration. The most difficult task that the project manager faced was how to engage all of the team members in active participation during all phases of the project. The lack of effective communication tools, a limiting organizational hierarchy, and trouble in maintaining enough momentum to meet schedule deadlines were all obstacles that the project managers had to overcome.

Effective team integration seemed to happen best when people are force to work together to produce a deliverable. The new model proposed in this document seeks to improve this by creating early deliverables that involve the entire team. The model also stipulates that daily synchronous communication is vital in making rapid

progress. Email is too slow of a medium for communication when collaborating on a deliverable.

The new model also proposes a flatter team organization with less well-defined roles. The responsibility of completing each phase of the project rests on the entire team, not just a small group of people. This way, each team member plays an active role in all aspects of the project. The new model also introduces a vice president role, filled by the instructor, who acts as an authority figure to evaluate performance and give feedback to each team member.

Finally, the new model modifies the traditional waterfall development process by establishing hard deadlines based on dates rather than progress for certain portions of the development cycle. In an academic environment, these deadlines work to create a sense of urgency which was lacking in the DISEL project. The scheduling of deadlines for detailed design and implementation is done after the scope of the project is known, enabling the team to make a more accurate assessment.

Many benefits may be reaped by moving to a distributed development environment. Engineering talent from other countries can be more effectively leveraged. Time zone differences can be taken advantage of to yield longer team workdays. The DISEL project succeeded in proving that distributed software engineering is a viable option. The problems and pitfalls encountered can be effectively managed by refining development models and creating better tools for person-to-person communication.

Appendix A

Team Contract

1. Communication Procedures

   (a) Tools

   - DISEL Web site is the basic structure. All the project documents will be there. This is the main way to organize the project.
   - The format for the documents is HTML
   - The Documentation Specialist will send a message to notify when a new document be placed in the Web.
   - The way to send the documents to the Documentation Specialist is:
     - The people at MIT will send the documents as e-mail attachments. They must specify which documents were sent and some specials consideration about those (like link for instance).
     - The people at CICESE must place the documents directly in the site. The place will depend of the document and it will be determine by the Documentation Specialist.
   - For document analysis will be used email. The emails must keep the subject of the mail that started the thread.
   - For general interchange of information will be used the email too. Each email must have in the subject one of the following word that indicate how urgent is the message:

     [IMMEDIATE REPLY] this mean that the sender needs a immediate response to the e-mail.

     [FYI] this mean that the email is just for information and the receiver don't need to replay the message.
   - Additionally, the sender must be clear about What, Who and When the response will be needed.

- For personal communication we will use tool like talk, Net Meeting or Chat.
- We will use the hypermail to keep a backup of the emails.

(b) Minimal feedback time

- The communication between groups should be at least once a week.

(c) Calling for meetings

- All agendas request must go through the V&V team before notify to the rest of the team. The V&V team will notify if some changes was made to the agenda.
- All changes and request for the agendas must be sent 48 hr. before the meeting to the V&V team. If the agenda is on the Web won't be modifications at least that the modification be urgent.
- The global meetings out of time of class will be notify previously. This meetings will be only few times.
- Each sub-team will decide their own meetings.

(d) Amount of preparation expected before a meeting

- Each one must read the agenda of the meetings al least one hour before the meeting.

(e) Information you expected in response to a query

- The response to a request must be at least a answer and when apply the response must include references about request (bibliography, link, etc.).

(f) Excusing yourself from a meeting

- If you have an appointment with some one and you can not attend, you must cancel the appointment al least 1 hour before.
- If you can not attend to a meeting you must notify to one of the Project Managers at least 24 hr. before the meeting.
- If you going to leave a meeting before finish it, you must notify to the moderator before the meeting. If there is an emergency, whispered it to the moderator.

(g) Hierarchy

- Teachers-¿ Project Manager-¿ all team.
- Teachers don't take decisions about any particular job of team members, if they need something, they must to request it to the Project Manager.

2. Conflict Manager.

(a) Decisions

- For general direction policies everyone needs to agree. For other decisions this will be decided voting If there is a deadlock the project manager must to decide. For more specialized decisions the role team is responsible for the decision.
- If there any complaints or issues between team members, it should be handled with the project manager in private.

Bibliography

[1] D. Ancona, T. Kochan, M. Scully, J. Van Maanen, and D. E. Westnety. Making teams work. In *Managing for the Future: Organizational Behavior and Processes*, module 3. South-Western College Pub., Cincinatti, Ohio, 1996.

[2] V.R. Basili. *Tutorial on Models and Metrics for Software Management and Engineering.* IEEE Press, 1980.

[3] B.W. Boehm. *Software Engineering Economics.* Prentice Hall, Englewood Cliffs, NJ, 1981.

[4] B.W. Boehm. Software engineering economics. *IEEE Transactions on Software Engineering*, 1984.

[5] F.P. Brooks. *Planning a Computer System.* McGraw-Hill, New York, 1962.

[6] F.P. Brooks. *The Mythical Man-Month.* Addision-Wesley, Reading, 1985.

[7] P. Coad and E. Yourdon. *Object-Oriented Design.* Prentice-Hall, 1991.

[8] M. Cusumano and R. Selby. *Microsoft Secrets.* Free Press, New York, 1995.

[9] M.E. Fagan. Design and code inspections to reduce errors in program development. *IBM Systems Journal*, 1976.

[10] T. Gilb and D. Graham. *Software Inspection.* Addison-Wesley, 1994.

[11] W.E. Humphrey. *Managing the Software Process.* Addison-Wesley, 1989.

[12] IEEE. *Software Engineering Standards.* IEEE Press, 1987.

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

[14] J. McCarthy. *Dynamics of Software Development.* Microsoft Press, Redmond, 1995.

[15] H. Mills. Chief programmer teams, principles, and procedures. *IBM Federal Systems Division Report FSC 71-5108*, 1971.

[16] D.L. Parnas. Information distribution aspects of design methodology. Technical report, Carnigie-Mellon University, 1971.

[17] C. Watson and C. Felix. A method of programming measurement and estimation. *IBM Systems Journal*, 1977.

[18] E. Yourdon and L. Constantine. *Structured Design.* Prentice-Hall, 1979.