

**An Empirical Study of the Coordination
In A Distributed Software Development Team**

By

Xusong Xie

B. Arch. (1998)
Tsinghua University

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

MASTER OF SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

© 2004 Massachusetts Institute of Technology. All rights reserved.
September 2004

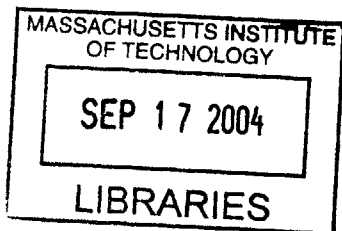
Signature of Author.....
Department of Civil and Environmental Engineering
August 6, 2004

Certified by.....
JoAnne Yates
Sloan Distinguished Professor of Management
Sloan School of Management

.....
Wanda Orlikowski
Eaton Peabody Chair of Communication Sciences & Professor of Information Technologies and
Organization Studies
Sloan School of Management

.....
Steven Lerman
MIT Class of 1922 Distinguished Professor of Civil and Environmental Engineering
Department of Civil and Environmental Engineering

Accepted by.....
Heidi Nepf
Chairman, Departmental Committee on Graduate Studies



BARKER

An Empirical Study of the Coordination
in a Distributed Software Development Team

by

Xusong Xie

Submitted to the Department of Civil and Environmental Engineering
on August 6, 2004 in partial fulfillment of the
requirements for the degree of Master of Science

ABSTRACT

As today's software systems become more and more complicated, coordinating the development of such systems has been an important factor to their successful implementation. The need for good coordination is especially important when the development team is geographically distributed and has to rely on information and communication technologies to support its activities. With limited available coordination mechanisms, distributed software teams need to carry out a set of coordination functions effectively throughout the software development process. In addition, in response to the changes in context and task, distributed software teams needed to be adaptive in their coordination. In this study, I try to understand how different coordination methods and tools could serve the changing coordination needs in software development through an empirical study of a distributed software team's practice.

Acknowledgement: I would like to thank the members of Little Company (LC) who generously provided the data and their time for this study. Special thanks also to Stephanie Woerner for her extensive research support on this project. This research was funded by the National Science Foundation grant number IIS-0085725.

Thesis co-supervisors:

JoAnne Yates
Sloan Distinguished Professor of Management
Sloan School of Management

Wanda Orlikowski
Eaton Peabody Chair of Communication Sciences & Professor of Information
Technologies and Organization Studies
Sloan School of Management

Thesis reader:

Steven Lerman
MIT Class of 1922 Distinguished Professor of Civil & Environmental Engineering
Department of Civil and Environmental Engineering

(This page is intentionally left blank)

ABSTRACT.....3

CONTENTS5

LIST of TABLES.....7

LIST of FIGURES.....8

CHAPTER 1 INTRODUCTION9

1.1. Background and Literature Review.....10

1.2. Site.....15

1.3. Research Methods.....16

CHAPTER 2 LITTLE COMPANY (LC)22

2.1. Group Context.....23

2.1.1. Group Characteristics.....23

2.1.2. Situation.....24

2.1.3. Task 28

2.1.4. Potential Conflicts29

2.1.5. Organizational Issues30

2.2. Group Processes.....31

2.2.1. Software Development Methods31

2.2.2. Task Focus32

2.2.3. Use of Time33

2.2.4. Participation35

2.2.5. Communication Patterns36

2.2.6. Flow of Activities37

2.2.7. Conflicts39

CHAPTER 3 RESEARCH FINDINGS40

3.1. Coordination Activities.....40

3.1.1. Monitoring.....41

3.1.2. Tracking and Control.....43

3.1.3. Notification.....46

3.1.4. Task Allocation.....48

3.1.5. Sequencing and Synchronization.....51

3.2. Coordination Mechanisms.....	54
3.2.1. Team Email List.....	55
3.2.2. Dyadic Emails.....	59
3.2.3. Conference Calls.....	61
3.2.4. Dyadic Phone Calls	65
3.2.5. CVS System.....	65
3.2.6. Norms.....	68
3.2.7. Documentation	70
3.3. Dynamics in Coordination.....	72
3.3.1. Dynamics in “Notification” Emails.....	75
3.3.2. Dynamics in “Coordination/Scheduling” Emails.....	77
3.3.3. Dynamics in “Internal Phasing” Emails.....	79
CHAPTER 4 CONCLUSION.....	83
APPENDIX A - Coding scheme used in analyzing LC’s email archive.....	86
APPENDIX B – A day in Dan’s life.....	88
REFERENCES.....	89

LIST of TABLES

Table 1.1 Characteristics of Teams: Traditional and Virtual.....	11
Table 1.2 Main coding scheme for analyzing emails.....	18
Table 2.1 Purposes of emails to team email list.....	31
Table 2.2 Participation of LC members in technical group discussions.....	35
Table 2.3 LC members' presence rates in conference calls.....	36
Table 3.1 Percentage of emails with the whole team as recipient (from 1997 to 1999)	56
Table 3.2 Dyadic emails between LC members (from 1997-1999)	60
Table 3.3 The use of coordination mechanisms in LC.....	72
Table 3.4 Timeline of technical activities in LC (1998-1999).....	74

LIST of FIGURES

Figure 2.1 Conceptual framework for collaborative work	22
Figure 2.2 Geographical distribution of LC members	24
Figure 2.3 Daily pattern of email communication in LC (in local time)	34
Figure 2.4 Weekly pattern of email communication in LC (in local time) from Jan 1997 to Feb 1998	34
Figure 3.1 Files checked in to the CVS system (1998-1999)	73
Figure 3.2 Comparison of files checked in to the CVS system and “Notification” emails (1998-1999)	75
Figure 3.3 The percentage of “Notification” emails in total emails (1998-1999)	76
Figure 3.4 Comparison of files checked in to the CVS system and “Coordination/Scheduling” emails (1998-1999)	78
Figure 3.5 The percentage of “Cooridnation/Scheduling” emails in total technical emails (1998-1999)	79
Figure 3.6 Number of “Internal Phasing” emails (1998-1999)	80
Figure 3.7 The percentage of “Internal Phasing” emails in total technical emails over time (1998-1999)	80
Figure 3.8 Changes of coordination emails (1998-1999)	82

Chapter 1

INTRODUCTION

Distributed software development has become common practice and an important part of companies' strategic efforts in searching for effective and efficient ways of developing new software systems. Flexible work arrangements and reduced operating costs are the most notable benefits of distributed software development. And such benefits are attractive to both large corporations and small start-ups.

However, the inherent characteristics of distributed software teams represent challenges as well as opportunities. The lack of rich communication channels, inadequate informal and personal interactions, and increased coordination burdens, among others, have together made it difficult to take full advantage of distributed work.

The coordination of a distributed software team has proved to be a crucial factor to the success of a project. The interdependencies among tasks and individuals require effective and efficient coordination to ensure that members function as a team and the resulting product functions as an integral system. The effective coordination of a distributed software team's activities, given all the restrictions and opportunities, has great implications for today's and tomorrow's software engineering practice.

This research is based on an empirical study of the coordination practices in a small distributed software company. This thesis is constructed in the following way: in the first chapter, I will discuss the background of this research, the research site, and my research

methods; then in Chapter 2 I will study how members of my research target - the LC company - functioned as a team in their situation; in Chapter 3 I will talk about three topics: how LC members met their coordination needs in the software development process, how they used different coordination mechanisms, and how the coordination activities evolved over time; finally, I will sum up in Chapter 4 and briefly discuss the implications of my research.

1.1 Background and Literature Review

As software systems become more and more costly and complex, people are seeking new ways of developing software systems to cut down cost and increase quality. With the rapid development in information and communication technology (ICT) and the increasing availability of related facilities, remote collaborations between two or more distributed sites become possible, enabling software project managers to have more options in structuring the teams and obtaining cost and productivity benefits, software engineers to enjoy more flexible work schedules, and companies to achieve strategic advantages such as reduced time-to-market, and proximity to customers etc. (Dewan et al., 2001; Carmel, 1999).

As a result, a new form of organization – the “virtual team” – emerges (Carmel, 1999). Table 1.1 shows the characteristics of traditional teams and virtual (or geographical distributed) teams. Today, more and more “virtual teams” are formed in different industries in order to take full advantage of distributed work. The software industry is no exception.

Traditional teams	Virtual Teams
Co-located members	Distributed members
Face-to-face interactions	Electronic communications
Hierarchical	Networked
Mostly informal communication	Continuous structured communication
Position authority	Process and knowledge authority
Information distribution (push)	Information access (pull)
Information on paper	Information electronic
Sharing completed work	Continuous sharing of incomplete work
Knowledge hoarding	Knowledge sharing
Transparent process	Computer-visible process
...	...

Table 1.1 Characteristics of Teams: Traditional and Virtual
(Source: Carmel, 1999)

The emergence of this new organizational form in the software industry brings about new challenges. Traditionally, people have known that there are many reasons that software projects get into trouble, such as scale, uncertainty, and interoperability (Pressman, 1997). Good project management is necessary to ensure both the quality and performance (on time and on budget) of the project. Traditionally, a lot of research has been done to develop new and enhanced methods and tools for improving software development performance (Fichman and Kemerer, 1993; Henderson and Cooperider, 1990). Recently, some researchers have started to study how to improve the practice of software project management based on the software development process. Topics such as task decomposition, task assignment, and work group coordination, as well as the social context (such as goal orientations and conflict) surrounding software developers' participation and interactions, have drawn a lot of attention in recent literature (Crowston, 1997; Hunton and Beeler, 1997; Kirsch, 1996; Waterson et al., 1997; Andres and Zmud, 2001).

One of the crucial factors influencing the success of distributed work is coordination. Any task that requires more than one person to finish needs explicit coordination, and any organization needs coordination among its constituents in order to operate (Van de Ven, et al., 1976) And this is true for many of today's software teams, which are developing complicated software systems that require knowledge in many domains and efforts of

many persons. The overhead of control and coordination associated with any software project is astounding. Developers spend as much as 70% of their time working with others (Demarco & Lister, 1987) and as much as 40% of their time waiting for resources or doing other work (Carmel, 1999; Perry et al., 1994). Successful coordination in software development ensures the effectiveness and efficiency of the project.

One of the well-known approaches to coordination is coordination theory (Malone & Crowston, 1990), which looked at coordination from a dependency perspective: “coordination is managing dependencies between activities.” Other researchers define coordination as the direction of “individuals’ efforts toward achieving common and explicitly recognized goals” (Blau & Scott, 1962) and “the integration or linking together of different parts of an organization to accomplish a collective set of tasks” (Van de Ven et al., 1976). Despite the different ways of describing coordination, most agree that the objective of coordination is to make people work together to accomplish some common tasks efficiently.

Distributed/dispersed teams create further burdens on coordination and control mechanisms. In geographically distributed software engineering practice, project tasks and team members are split across different sites, resulting in the loss of many rich interactions and close coordination. People cannot coordinate by peeking around the cubicle wall, nor can managers control by strolling down the hall and visiting team members’ offices. In addition, as Fielding et al. (1998) pointed out, people in virtual enterprises cannot observe and anticipate the factors that affect the interdependencies among tasks. Finally, as coordination needs for global software teams rise, so too does the load on all forms of communication – from the telephone to less obvious communication channels, such as multi-site project management software (Carmel, 1999).

According to many researchers (Crowston, 1997; Hunton & Beeler, 1997; Kirsch 1996; Kraut & Streeter, 1995; Waterson et al. 1997; Andres & Zmud, 2001), task and the social context of a project together define the coordination strategy. “Project coordination strategies must exhibit communication mechanisms and decision-making structures that match or fit the task and social context associated with specific work units and project

phases” (Andres & Zmud, 2001). Depending on the particular situations and the characteristics of the project, distributed software teams may take many different structures and operate in very different environments.

To better understand the coordination practices in distributed teams, it is helpful and necessary to understand how they function as groups. According to Homans (1950), a group is defined by its members’ interactions and consists of its members’ activities, interactions, sentiments, and the mutual relationships existing among these elements. In addition, Simon (1957) suggests that group interaction depends on the group’s tasks, the previous levels of interaction resulting from the levels of relations among group members, and the existing activities within the group. More recently, Tubbs (2001) stated that a group’s size and structure might additionally define it. The role of each member in the team as well as the whole hierarchy greatly affects the interactions within the group (Joseph et al. 2003). Other researchers also revealed that the context of the group has great impacts on the group process and the group members’ behaviors (Olson & Olson, 2001).

What kinds of coordination challenges do distributed software teams face? In distributed software teams, the coordination needs share a lot of similarities with those in traditional co-located teams. For example, basic coordination needs such as planning and scheduling the technical milestones still remain an important issue in distributed software development. Nonetheless, the specific needs might vary depending on the situations. In some instances, the differences are quite big and require careful consideration. For example, member’s awareness of the group’s status and activities has been considered by many researchers to be an important aspect in making the team’s operation smooth, and a lot of research has been carried out to support group awareness in collaborative work (e.g., Cadiz et al., 2002; Gutwin et al., 1999; Rodden, 1996; Jang et al., 2000).

Developing adequate coordination mechanisms is a central challenge for distributed software teams. According to Simone (1995), “a coordination mechanism (CM) can be defined as a protocol, encompassing a set of explicit conventions and prescribed procedures and supported by a symbolic artifact with a standardized format, which

stipulates and mediates the articulation of distributed activities so as to reduce the complexity of articulating distributed activities of large cooperative ensembles.” In software engineering, such coordination mechanisms include milestone schedules, project documents, “coffee break chat”, and face-to-face group meetings, among others. The availability of coordination mechanisms is more limited in distributed software teams. It is very important that teams selectively revise traditional coordination mechanisms and create new ones based on their specific coordination needs in their practice. Much research has been carried out to study the effectiveness of different coordination techniques such as project documentation and group meetings in traditional software teams (e.g. Kraut et al., 1995). It is notable that informal coordination mechanisms, which seem to play an important role in traditional co-located teams, are difficult to maintain in distributed software development. How to take better advantage of both formal and informal coordination mechanisms remains an important issue in distributed software engineering practices.

Another interesting and important topic in software development is the dynamics of coordination. Most current research (e.g., Kraut et al., 1995) examines the effectiveness of coordination mechanisms based on a static viewpoint, that is, they study the use of such coordination in a specific stage (e.g., testing, requirement analysis) of the software development cycle, without considering dynamic project factors such as system growth, system stability, etc. Dynamic factors are important because they could lead to differences in the intensity (frequency) of coordination needed at different stages of system construction (Chiang and Mookerjee, 2002). What is more, these dynamic factors could also create different coordination needs as the project evolves.

I believe that we could better coordinate software development activities if we could better understand what kinds of coordination activities are needed most for specific kinds of development activities. Moreover, coordination support for software development activities should be adaptive to the dynamic changes of activities throughout the software development cycle. This thesis will also try to study the patterns of coordination activities and their relations to development activities.

1.2 Site

Little Company (LC) is a small, primarily self-funded, software start-up company. During the period under study, LC was building a new programming language product, the LC system (names of the company, product, technology, and members have been disguised for confidentiality reasons). They planned to take advantage of an emerging new technology and develop a product that would have better performance and probably shorter time-to-market than similar products developed by larger competitors.

The company was originally formed by four computer experts, and was joined by another one soon after. Of these five members (four of whom hold doctoral degrees), three (Keith, Robert, and Dan) worked full time for LC while the other two (Fred and Martin) worked part time and dropped out about two years later.

During the time LC operated, its members were working in their individual home offices in four cities in three time zones across the US. Each LC member was working on a relatively flexible schedule. There were no clear separations between work time and life time, and their work was often interrupted by or intertwined with personal issues and other non-work-related events. They often worked overtime into the night and/or during the weekends while dealing with personal matters during regular week day hours.

LC members did their development work on their local computers. Finished and tested components were periodically sent through the Internet to a central server where they were merged together into one single system. And all LC members did an update from this central server from time to time to make sure they were working on the latest version of the product.

In LC's software development method, they first developed a functional system with most of the required components and functions, and then they continuously made changes to the product to fix bugs, improve performance, and add/change features to make a better product. Customer feedback, third-party evaluations, and competing products often served as the motivation for such changes. The first functional product

was developed one year after the company was formed, while the team continued to work on it in the following three years.

To communicate with each other and coordinate with each other's development activities, as well as to run the business, LC members relied on several communication methods. Email was used extensively and served as one of the main communication channels among LC members. Telephone calls, including weekly conference calls with all members, were the other major way to exchange information within the company, report work status, and plan and schedule development work. Face-to-face meetings between LC members were rare due to multiple reasons such as geographical distance and budget constraints. Other Internet-based technologies such as web-based video conferencing, audio conferencing, or instant messaging were not used partially because they were not yet widely available at the time the company was formed (detailed discussion of the communication issues within LC is in Chapter 3).

1.3 Research Methods

In this study, I try to understand the practices of LC as a virtual software development team and the context in which LC operated. Based on this, I then study the coordination functions and mechanisms LC members used during the development of the LC system.

This analysis will focus on qualitative aspects of the company's work practice. Specifically, I will study and evaluate LC's use of the team email list, the conference calls, the dyadic emails and phone calls, and the CVS (Concurrent Version System) system as coordination mechanisms in distributed software development.

The data sources I used in this research come from the email archives, the activity log of the CVS server used for code management, the conference call minutes kept by a member, and interviews with four of the company members.

Email

One of the full-time LC members (Dan) kept all the received emails related to the LC project and the company from the very beginning. We got all these emails from Dan and

supplemented them with additional messages saved by the other two full-time members. The final email archive was the main data source for our detailed analysis of LC members' activities and communications during the time period. Although it didn't include all of the dyadic emails between some members (e.g., dyadic emails between Fred and Martin are not included), we still believe it captured most of the activities within LC during the time period due to the fact that LC members usually included the whole team in their email discussions (even when the topic only directly affected some of the members).

The email archive included emails from early 1996 when some members of LC began to organize the company, to late 2000. In this study, I chose to study the data from early 1997, when all LC members were in place and the LC project began, until late 1999 after the LC system was released to the public.

As part of a larger research project, we used a coding scheme to study the email archives for better understanding of the interactions between and among the team members. We read through all the emails and coded them using a coding scheme designed so that it categorized all the emails according to their purposes (why), content (what), timing (when), and work practices involved (how). Some of the sub-categories (such as coordination/scheduling) were coordination-related and provided important information for this study. (See Table 1.2; for a detailed description of each category, please see the Appendix A).

CATEGORIES
A. Purpose (Why)
A1. Initiating discussion
A2. Coordinating/scheduling
A3. Discussion
A4. Report
A5. Notification
A6. Other
B. Content (What)
B1. Technical
B2. Administrative
B3. Personal
B4. Other
C. Timing (When)
C1. Internal phasing
C2. External deadlines
C3. Other
D. Work Practice (How)
D1. Reference to media switches
D2. Instant messaging
D3. Reference to problems
D4. Requesting help
D5. Offering help
D6. Disagreement
D7. Reference to external expertise/network
D8. Negotiation (internal)
D9. Identity
D10. Other

Table 1.2 Main coding scheme for analyzing emails

CVS log

LC used a central server to store the code of the LC system and other company/project related information such as business documents. The server was owned and maintained by an ISP (Internet Service Provider) company, and all LC members could access the server through the Internet to upload (check in) or download (check out) information. The server kept track of all the check-ins by maintaining a record of the file information, version of the file, date/time, size of modifications (number of lines added and removed), and the modifier's comments. Most of the check-ins involved multiple files in several different directories.

Following is a CVS log record example about a check-in of a file onto the server:

```
“revision 1.12  
date: 1998/01/15 18:58:48; author: radar; state: Exp; lines: +13 -11  
Repairs to choice expression code generation, silenced IO debugging.”
```

The CVS log contains the following useful information¹:

Revision: The version of the file committed. In the example, the revision is 1.12.

Date: The date/time at which the commit was executed.

Lines: The number of lines of code added or removed in the new version.² In the above example, 13 lines of new code were added and eleven old lines removed.

Comments: Brief note from the author about the new change in the submission.

The CVS was set up and put into use in late 1997 and was still running at the time we did this research. In this study, I limit the research to data before late 1999, corresponding to the time period for which I studied the email archives.

Conference call minutes

LC maintained regular weekly conference call meetings among all its members during the period studied. This was an essential way to maintain the team relationship and make sure every member was making good progress towards the team’s goal. LC members usually used such conference calls for discussing and making important business decisions, coordinating development activities, and assigning tasks to each member.

¹ Unfortunately, the CVS server didn’t distinguish among different submitters and recorded all of them as “radar.”

² For text files (such as source code files on the server), the number of lines added/removed is generally a good measure of the magnitude of the changes. But for files in binary format (e.g., pictures), the situation is more complex and the number of lines added/removed is not a direct indication. In this research, since we are only interested in the technical development work, I only consider the source code files, which were text-based.

Robert served as the company secretary of LC and took “minutes” of every conference call. The original transcript was hand-written but later it was transcribed into digital format. These minutes include information about the date/time, people present (or absent), and the key points discussed during the call. Such minutes were intended to serve as a memo for later reference if necessary, but were not regularly distributed to the other members.

I analyzed the conference minutes to study the patterns of using conference calls to coordinate each team member’s activities.

Phone call logs

Besides holding the regular conference calls, LC members also used two-way or three-way calling extensively in their daily work. Unlike the conference calls, the two-way or three-way phone calls were not recorded or summarized in written form so we do not know their content. However, we do have access to the logs of the phone calls between LC members from mid 1997 to Dec 1999. The logs provide information about the date/time/duration, caller, and receiver(s).

Interviews

During the larger research project, several interviews with LC members were carried out to better understand their practices. Different researchers conducted the interviews at different times and using different methods. For example, in early 2000, two researchers conducted one day of interviews with three of the LC members -- Dan, Keith, and Robert – both individually and as a group. The interviewers asked questions about the history of the company, team members’ backgrounds, their work practices, technologies used in their work, their attitude towards their work patterns, etc. Three years later, three researchers interviewed Fred, with one researcher calling in on speaker phone. Several other researchers conducted follow-up interviews with Dan during weekends or off time.

Most of the time such interviews were recorded by a digital recorder and were later transferred into computers. A third-party professional secretarial services company made transcripts of some interviews. I have been careful in using such transcripts, sometimes

comparing them with the original audio records to avoid possible errors and misunderstandings.

Due to the different time spans of our data sources, I was not able to do full-period research with all needed data. However, for a specific time window (from the beginning of 1998 to the end of 1999), all four types of time-relevant data (email, phone logs, CVS logs, and conference minutes) are available. This specific time window covered many of the important stages of the product development process. I believe the available data is adequate for studying the coordination practices of a distributed software development process, especially in the implementation-test-revise stages, and that the results of this research have general implications for the overall software development cycle and for other small software development teams.

Chapter II

LITTLE COMPANY (LC)

In the following sections, I will use Olson & Olson (2001)'s framework for collaborative work in discussing the LC team members, their development work, and the context. Specifically, I will examine how the group context affected the way LC operated as a team, focusing on the main boxes of this diagram and considering only the subtopics that are relevant to LC. After considering the group's context (boxes on the left) I will consider group processes (middle box).

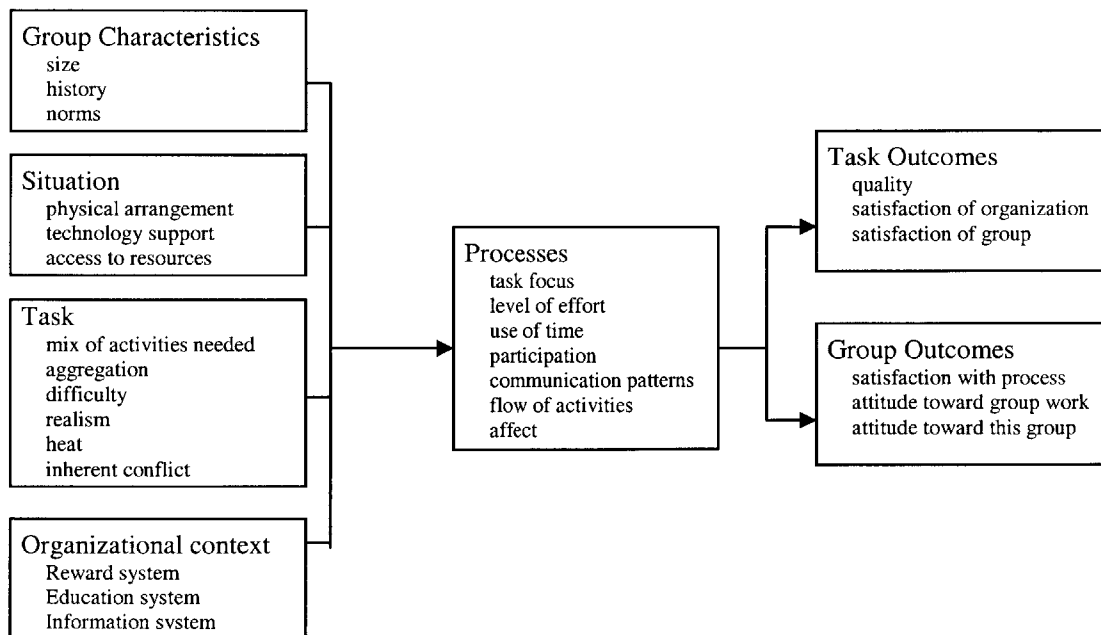


Figure 2.1 Conceptual framework for collaborative work
(Source: Olson & Olson, 2001)

2.1 Group Context

In this section, I will discuss the group characteristics, situation, task, potential conflicts, and organizational issues of the LC team.

2.1.1 Group characteristics

Like many start-up companies, LC had limited available resources including human resources: it had only three full-time employees and two other members working on a part-time basis³. The two part-time members stopped playing active roles in LC in 1999.

LC members had existing personal relationships before the beginning of the company: Dan, Keith, and Fred got their Ph.D. degrees from the same department of the same university; Dan and Keith were friends and had experience writing papers together at a distance; Keith and Robert had become friends while working in the same large company; and Fred and Martin also had experience working together and lived in the same area and met regularly. Among the five members, Dan met Robert before LC was started, and neither of these two ever met Martin before, during, or since they worked together at LC. Martin and Keith met for the first time a year or so into the project.

The existing personal relationships among the LC members played a critical role in putting the team together and allowed them to work closely in LC where they didn't have many opportunities to see one another. Research shows that trust has a large effect on the successfulness of teams, but that trust is relatively difficult to establish in distributed teams (Layzell et al., 2000). The trust in each other from existing relationships among LC members ensured the determination and involvement of each team member, especially at the early stages of the company. The established trust among particularly the full-time LC members continued to be enhanced in the course of the project and played an important role in running the business, as well as in dealing with conflicts within the company.

2.1.2 Situation

³ For more information about the size effect on a team, see Bradner, Mark, and Hertel 2003.

The situation in which LC members worked included geographical distribution, technology support, and access to resources. LC members worked in their personal homes or private offices located across the US: Dan and Keith in two nearby states on the east coast, Robert in the Mountain Time Zone, and Fred and Martin in the same metropolitan area on the west coast. Due to the distances between sites, LC members never had an all-member face-to-face meeting during the time they worked for LC. However, some members did have more chances to see each other due to their relative geographical proximity.

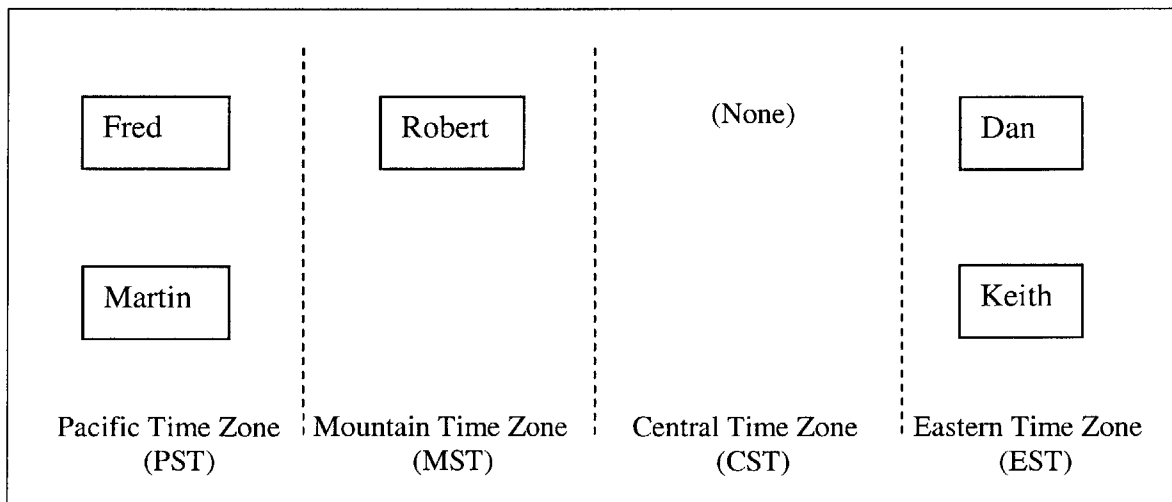


Figure 2.2 Geographical distribution of LC members

LC's physical arrangement had important implications for the way LC members organized their individual work and interdependencies.

-Lack of informal face-to-face meetings. LC members worked in home offices that were relatively isolated from each other. For example, they were not able to “overhear” or “oversee” each other’s activities by sitting close to each other’s desk, and they were not able to have a short exchange of ideas when running into each other in the hallway. According to Mills (1967) and Hause et al. (2001), the lost of such rich interactions would be expected to have negative impacts on the way they worked as a team, because of both the increased coordination cost and the weak interpersonal relationships that are crucial to maintain a team.

-Different time zones. Besides the physical separation, LC members also faced differences in time zones and work patterns. Dan and Keith lived in the Eastern Time Zone, while Robert lived in the Mountain Time Zone, which was two hours later, and Martin and Fred lived in the Pacific Time Zone, three hours later than Dan and Keith and one hour later than Robert. Such time zone differences required LC members to keep the time differences in mind when they needed to schedule some group activities, or when they needed to interact with each other in a timely manner. Potential problems might include wasted development time or interrupted personal life.

Given this geographical dispersion, LC members needed to select appropriate technology supports for both their development work and their communications.

LC members did their development using PCs in their home offices. They tried to keep their computers up-to-date by upgrading them from time to time. Generally speaking, their hardware was able to meet the requirements of their development activities. As Dan noted it in a personal communication (July 31st, 2004):

“Strictly speaking, though we did not have access to large servers like we would in a large company, our desktop computers were substantially more capable than a typical desktop at other large companies that we know of. This is partly because our desktops did double-duty as servers.”

LC also had a CVS server which served both as the central code storage space and the change management system. The CVS server played a central role in LC members’ distributed software development activities. The CVS system allowed LC members to do their individual development work, without having to worrying about new changes by other members, until they finished up their own work, committed it onto the server, and merged it with other parts of the LC system. The CVS system also provided a way for LC members to track the changes made to the LC system and allowed them to trace possible causes of bugs or undo changes that had caused problems. The CVS server was hosted and serviced by an Internet Service Provider.

At the time LC was formed, Internet access was available to general households on a dial-up network that utilized phone lines for data communication and connected computers to the Internet. The bandwidth of dial-up was pretty low (usually ranging from 33.6kbps to 56kbps) and sometimes unreliable. What is more, the phone could not be used for either incoming or outgoing calls when the dial-up network was in use. All three full time members switched to a more advanced Internet access technology (Robert in 1999, and Dan and Keith more than a year later) - DSL (Digital Subscriber Line) - which provided higher Internet access bandwidth (about 1.6Mbps) while at the same time allowing for phone calls.

In terms of development tools, LC members were using a software development kit (SDK) which was broadly employed in the industry and had been evolving during the whole LC system development cycle. The SDK was a commercial product and was easily available in the market with customer support. LC members had been getting help from the customer support department with problems related to the SDK.

As for communications, the dispersed LC members needed to rely almost completely on distance-communication technologies rather than on traditional face-to-face interactions. The availability of Internet access and computers allowed LC members to use email in their work. Besides that, traditional tools such as the telephone also served as a convenient way for frequent ad hoc conversations and an emergency contact channel.

During the development of the LC system, LC members used email extensively as a tool to communicate and coordinate. No standard email tool was required, and each LC member used his own choice. Sometimes LC members also used email to send small files as attachments to each other. A team mailing list was established as an efficient way of making announcements, sending out update notifications, and having group discussions.

Phone calls were widely used in LC because they allowed synchronous communications and were ubiquitously available. Not all their home offices had two or more phone lines, so normal phone calls had to share the phone line with Internet dial-ups. With DSL installed, it was possible to have normal phone calls while the dial-up network was in use.

LC didn't use video conferencing in their work, though this might had been a great help to them when face-to-face meetings were generally impossible. Traditional video conferencing required expensive equipment and the usage cost was too high. Newer technologies, such as Internet-based video conferencing systems, were not very reliable yet and also required expensive high-speed Internet services.

Access to resources was an important piece of the team's situation. During the software development cycle, different resources were required in order to meet the needs of the development effort. Specifically, besides financial resources, the most important resources for typical software development projects are computing resources (including hardware and software), human resources, and time.

The computing environment that supports the software project incorporates hardware and software. Hardware provides an environment that supports the development tools (software) required to produce the final product – the targeted software system. Because in most software organizations multiple constituencies require access to the computing resources, careful allocation of the limited resources is crucial in assuring the smooth operations of the whole team (Pressman, 1997). In LC, each member had a set of computing facilities (computer, software, network, etc.) in his home office for doing his own development work. These facilities usually met most of the local development needs. However, in order to commit their code onto the server and share it with other members, LC members needed to make sure they could use the CVS server without potential conflicts. Fortunately, because the CVS server itself provided a whole set of mechanisms to coordinate conflicting server usage requests, sharing the CVS server didn't constitute a serious problem in the development. CVS allowed for multiple individuals to read the code at the same time when nobody was writing to it. Whenever any user was writing to the code, the CVS prevented all others from writing or reading. This mechanism ensured the integrity of shared source code and handled potential conflicts.

Due to its limited financial resources, LC had to rely on its three full-time employees to take care of most of the administrative and business issues. They got some help from outsiders in the form of technical discussions and personal help. LC didn't have a

supporting staff, so every member had to take care of his own administrative issues. The three full-time members worked roughly 50 hours per week on the project. However, they had to split their time between development work and business/administration work. For example, Dan worked with the technical part of client engagement, and Robert served as the financial officer of the company, while Keith spent a lot of time on administrative matters. LC members tried to make better use of the available time resources by frequently reallocating tasks among members.

2.1.3 Task

During the development of the LC system, LC members needed to deal with a lot of tasks, both technical and non-technical. Though all LC members were technical experts, they still needed to be careful when working in a distributed environment.

The LC system is a complex systems tool. Its requirements underwent a couple of major changes to reflect the new progress of the technology, improve the performance, and incorporate new features. The technology was new and the industry standard was evolving at the time of the development, both features that added to the difficulties. Due to the existing competing products, the performance requirements of the LC system increased in the later development stages.

LC members had expertise in the product and had experience using technologies similar to what was used in the LC system. However, the fact that the technology was new and still evolving and that the industry standards for the technology had been ever changing created difficulties in developing the product. Uncertainties in both the computing environment and the technology itself proved to be one of the biggest problems in developing the LC system.

Besides purely technical tasks such as coding and testing, project management and support functions such as coordination, control, and logistics all represented demanding issues and might affect the technical development work. LC members were less familiar with the administration and business side of the company than with the technology itself.

2.1.4 Potential conflicts

The interdependencies of the LC system's components and the task allocation among geographically distributed team members constituted the most notable potential conflicts. A good understanding of the product's architecture allowed LC members to allocate tasks so that each member's roles and responsibilities were clear and known to every member. In subsequent sections, I will discuss these issues in detail.

Other potential conflicts in LC included:

- The interleaving of technical tasks and business/administration tasks made management complicated. The fact that none of the LC members had enough experience running a company forced them, especially the three fulltime members, to split their time between technical and non-technical work, making coordination and scheduling difficult.
- Among the LC members, only two members had worked together before, and another two had written papers together. Different work styles and cultures were a potential source of conflicts. Potential conflicts also existed between those who had worked together but failed to adapt to the new environment. Each team member's awareness of such differences and willingness to accommodate such differences would be the key to avoid such conflicts.
- LC members worked in home offices and this brought another potential source of conflicts: the work and personal life conflicts. The fact that LC members usually dealt with intertwining work and personal issues such as housecleaning and childcare increased the difficulties of ensuring the team's smooth operation.
- LC members' different preferences in communication media were also a potential source of conflicts that needed to be dealt with, especially when they were dispersed and relied almost exclusively on mediated communication for their daily work.

2.1.5 Organizational issues

To deal with the constraints on human resources and take advantage of outside expertise, LC members retained academic and industrial relationships with outside experts to keep pace with the latest technology progress and industry news. If an LC member found

interesting industry news or helpful information outside the company, he would inform the company by an email. Inside LC, members also learned from each other from the discussions of bugs or other technical issues. When they encountered technical problems or some other type of work in which none of the members had enough expertise, one member would then have to learn to deal with it and become the “go-to” person for the new problem. Xu (2004) describes such an example:

One of the essential modules in the system was linker, an integrative module to connect other components of the system during runtime. Keith, Dan and Robert were equally familiar with it based on their previous experience. It happened that Robert started working on linker in 1997. Gradually he became most knowledgeable about how this module behaved. During the later testing and debugging phase, when they detected a linker problem, it would be passed on to Robert, who was now considered the team’s expert on that. Dan talked about this in the context of debugging:

When we said it was a linker problem, it ended up in Robert’s lap because he worked on the linker to begin with and that’s what he knew the best.

LC employed an open information system within the company. Since all members were stakeholders in the company, almost all technical, business, and administrative information was made available to all members. In addition to discussing company issues in the weekly conference calls, they also used the team email-list for discussion, notification, and information distributions (See Table 2.1). They did this by putting the team email-list (or all members’ email addresses) in the “To” or “CC” list in their emails, sometimes even when the issue was addressed to a specific member. A lot of two-way and three-way phone conversations also went on between team members to address complicated issues, in addition to the weekly all-member conference calls where they reported their current work status and plans.

Purposes	Number of emails	Percentage of total team emails (6393)
A1. Initiating Discussion	1711	26.76%
A2. Coordinating/Scheduling	737	11.53%
A3. Discussion	2438	38.14%
A4. Report	30	0.5%
A5. Notification	1475	23.07%
A6. Other	0	0 %
Note: emails to team list comprise 64% of all the 9943 emails from 1997-1999		

Table 2.1 Purposes of emails to team email list

2.2 Group Processes

The processes LC employed in their practice were greatly affected by the context of the project team. In this section, I will examine the way LC members operated as a team in the development of the LC system.

2.2.1 Software development methods

In software engineering, the work can be generally categorized into three generic phases: the definition phase, the development phase, and the maintenance phase (Pressman, 1997). And the detailed activities within each phase would vary according to application data, project size, or complexity. In the time period studied, LC was mainly in the development phase, in which they needed to decide how to structure the data, how to implement the data as software architecture, how to implement the procedural details, how to characterize interfaces, how to implement the design into a real system, and how to perform the testing. Three specific technical tasks always occur in the development phase: design, implementation, and testing.

The software development process is a very important part of software production because it affects the overall efficiency with which different resources are used to produce the software. Software models allow us to describe the process of creating a product from the very beginning until its release. In practice, there are some models widely used in today's software engineering practice, including the Incremental Model, the Waterfall Model, the Spiral Model, the Rapid Application Development Model, and the Prototyping Model (Pressman, 1997)

Because of the complicated product and the uncertain market, LC members had to be adaptive and creative in using software development processes so that they could handle all kinds of changes.

Initially, LC's process resembled a traditional "Waterfall" software development methodology to develop their first functional prototype. That is, they identified the needs the LC system would fulfill and its advantage in the industry, analyzed and discussed the technical requirements, designed the system architecture, did the coding and testing, and came out with a system prototype that provided the fundamental functionalities in early 1998. The benefit of this was that they could have something functional ready within a short amount of time, without having to worry much about uncertainties in industry standards and various client needs.

Then their work focus shifted to revising and refining the product in the following two to three years, as the technology standards became more stable and the market needs became clearer. Their work process in this stage was similar to the Spiral Model (Pressman, 1997), in which they released multiple versions of the LC system and did problem identification, requirements analysis, implementation, and testing for each release. During this stage, the development work mainly addressed issues in certain aspects of the system rather than redesigning and reworking the LC system as a whole.

2.2.2 Task focus

The team initially focused its work on development and testing of the code. Finding and fixing bugs became routine work after every release and constituted an important part of LC members' daily work. Besides finding bugs through internal testing, LC members learned of problems that were reported by test users (usually an outside expert or a potential customer). Addressing these became an important part of the work in later stages, after public releases were made available to outsiders.

Adding important features to the system also required the team to monitor the trends in the industry as well as any changes in competing products, analyze and discuss each

candidate for additional features of the LC system, and implement the selected features as well as revising the existing system to accommodate such new changes.

Improving the performance of the system was also one of the high-priorities in the team's schedule. Due to high user requirements and competing products, increasing the system's capacity and improving its performance had been the team's long time goal. Usually such improvements involved changes in the design and implementation of several aspects of the LC system, and required the coordination and cooperation of the team members.

For some time in the development cycle, business tasks such as writing the business plan also took up a lot of time and effort and presented disruptions to the technical development activities. The business activities are beyond the scope of this research and will not be discussed in detail in this thesis.

2.2.3 Use of time

Though LC didn't establish official rules concerning regular daily work hours, the team members tried to maintain their own regular daily schedules. However, working at home was often interrupted by events such as housekeeping and personal phone calls. To maintain a certain number of work hours a day, LC members sometimes worked at night to make up for the disrupted work during the day as shown in Figure 2.3. It was also common for LC members to be involved in development work and other LC-related tasks during the weekends. Team members could use the weekend time to make up the time lost for personal reasons during the week, or use the extra time to finish some work that was late (see Figure 2.4). As Dan noted,

“...my time is broken up... I often don't get started until 10. Some days a week, I can work cleanly until five, but especially during the school year, we've had kid problems... Picking people up and running them around and doing stuff. Then, I would work in the evenings and sometimes I would work part of the day Saturday and part of the day Sunday.”

Extra work time during weekends was usually “at will.” LC members were not expected to be available for work purposes during weekends. So any member who decided to work extra time during weekends would have understood that he might not be able to get

information or help from other members enjoying weekends and should plan his work accordingly.

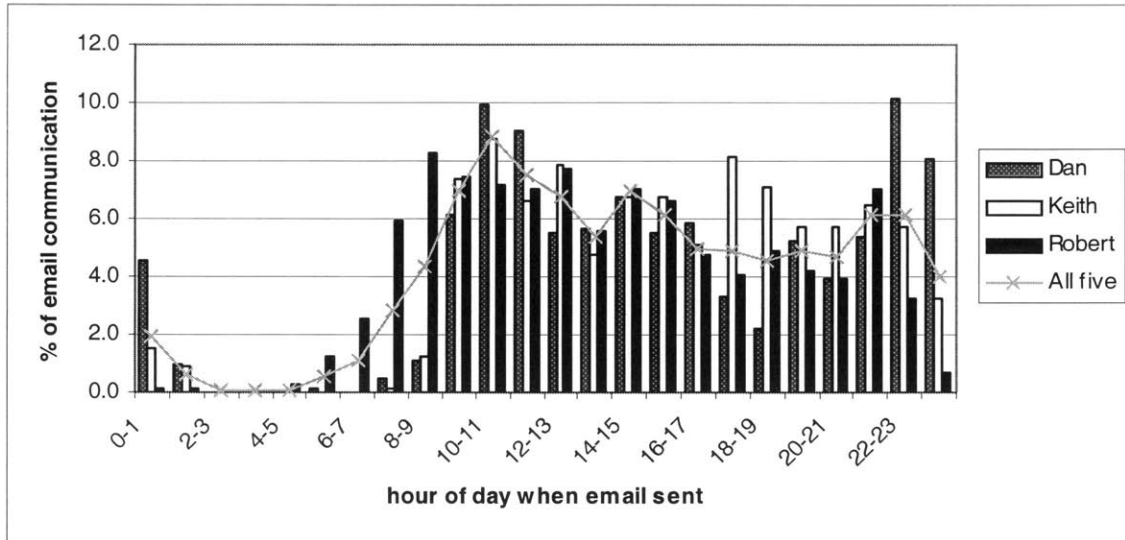


Figure 2.3 Daily pattern of email communication in LC (in local time) from Jan 1997 to Feb 1998. (Source: Im, Yates, & Orlikowski, 2004)

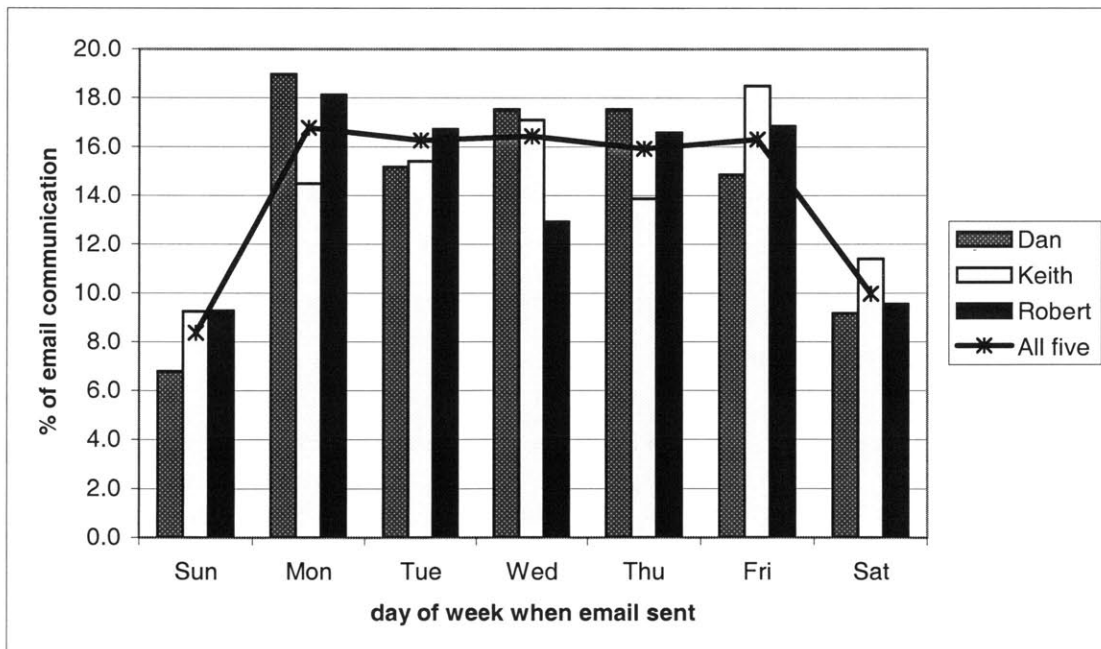


Figure 2.4 Weekly pattern of email communication in LC (in local time) from Jan 1997 to Feb 1998. (Source: Im, Yates, & Orlikowski, 2004)

2.2.4 Participation

LC operated in such a way that almost all information and all decisions were open to all members' participation. For example, LC relied heavily on the team email list to discuss the possible reasons and potential fixes for bugs found in their code. Each member was expected to contribute his knowledge and expertise in the group discussions of technical issues such as system design and bug-fixing. Due to the different involvements of full-time and part-time members, I observed from the email archives that Keith, Dan, and Robert participated much more in the technical group discussions than Fred and Martin did. For all the company-wide⁴ technical emails (coded as technical in category B), there were substantial differences in the full-time members' participation and that of the two part-time members (see Table 2.2).

FROM	Number of emails	Percentage
Dan	2625	33.71%
Keith	1977	25.39%
Robert	2423	31.12%
Martin	465	5.97%
Fred	243	3.12%
(Others)	54	0.69%
Total	7787	100.00%

Table 2.2 Participation of LC members in technical group discussions

The weekly conference call was the most important way to maintain the team cohesiveness among LC members as well as the venue for progress reports, task allocation, and the handling of other project/business management issues. Each member's regular and active participation in these conference calls was critical for the effective operation of the project team. And many important decisions, both technical and business, were made in such conference calls with all or most of the LC members presented. As I observed from the email communications about the conference calls as well as the minutes kept by Robert, LC members usually tried to make sure every member could participate in the conference calls. When some member could not make the conference

⁴ Emails with all LC members, or all LC members except the sender, as the recipients (on either the "TO" list or the "CC" list).

call at the regular time due to conflicts with personal or other business issues, they usually discussed this beforehand and changed the date or time of the regular calls to accommodate everybody's schedule. The three full-time members had a higher presence rate in the conference calls than the two part-time members did, reflecting their greater centrality to the team (see Table 2.3).

Member	Absence from meeting	Attendance percentage as of 135 meeting records (from Jan 1997 to July 7 1999)
Dan	2	98.5%
Keith	0	100%
Robert	0	100%
Fred	8	94.0%
Martin	24	82.1%

Table 2.3 LC members' presence rates in conference calls⁵

2.2.5 Communication patterns⁶

Communication played an indispensable role in the product development and business operations of LC. Since face-to-face interactions were mostly unavailable, the interpersonal interactions between LC members had to go through new and traditional remote communication channels.

Email was used extensively in LC members' daily work, because of its easy use, quick delivery, and low cost. Since most of the time LC members were working on their computers, whenever a code-related issue came up (such as a bug, or an unspecified function in the code), LC members could easily open an email client software, key in a few lines into a new email and send it out to another team member or the whole team for help. Another advantage of using email to discuss technical problems was that relevant information such as system error messages or the problematic code could easily be copied and pasted into the email, providing necessary information to deal with the issue. As Robert noted,

“...the advantage of email ... [to someone] like me who saves everything is that once you've put it in email, you've got some piece of documentation that you can go back to,

⁵ Data cover meeting from the first meeting in which all five members came together, until Martin stopped playing an active role in LC.

⁶ For more discussion on LC communication, see Ghosh et al., 2004.

whereas a telephone conversation, you forget what the aspects of it were that were important.”

While email was used mostly for asynchronous communications and occasionally for near-synchronous communications, LC members preferred traditional phone calls when time-sensitive issues were involved and immediate attention of another team member(s) was important. Examples of such issues included system crashes or bugs that made the development work come to a stop, and that could not be solved by one person alone. Again, Robert explained,

“...more than often, I will send an email message if I have a problem where I have like a stack trace on my screen of something that didn’t work right and died, I will put that into email and I will send it out, usually to everybody in the company.”

Another situation where phone was more frequently used was when complicated issues were being discussed and a lot of communications were expected to happen in a short time. Phone calls allowed for more real-time information exchange so people could quickly reach a common understanding of the issue and then work together to find a solution for it.

It is notable that LC members often used emails and phone calls together. Part of the reason was adaptation to the members’ different preferences on communication methods. Another situation leading to the use of two media together occurred when email was used as a notice to inform others about a forthcoming phone call. Robert explained LC members’ different preferences like this:

“... (Keith) wants to pick up the phone and talk to you. He leaves telephone messages that say call me. I would prefer that I get an email message saying call me about such-and-such or here’s the problem, three sentences, call me and let’s figure out what to do with this. ... if I want to ask (Dan) a question, I will often send him an email because I know there’s certain periods of times when he’s giving kids baths, he doesn’t want to talk, he’s doing dinner... and I may call and leave a telephone message if it’s something that I really can’t write down very well.”

Fax and traditional mail were usually used for delivering official documents rather than for general communication purposes.

2.2.6 Flow of activities

After LC members collectively worked on the requirements analysis and system design stages, most of the development work was divided into three main components – known by LC members as the front end, the back end, and the middle end -- and then allocated to individual members. Each member then developed different components according to the system design in his individual work place, normally his home office.

A lot of email exchanges concerned issues unforeseen during the previous analysis and design or in-depth negotiation of interface requirements between different components. As development work went down to implementation, ambiguous points in the design and implementation of the system emerged and were solved by the team together.

While working on his own code, each LC member also downloaded other system components that other members had submitted to a central code server. These other components would allow the developer to test his own code and see how it functioned within the system and how it worked with other components. Whenever a problem was observed, the developer would try to identify and locate its source. If the problem was within his code, the developer would find a way to fix it. If the problem existed in other components, the developer would then inform the owner of the problematic code and they would collectively decide how to fix it.

After the local development work met all the requirements to combine with other components and work, a member would do a component test to remove every bug he could find to avoid bringing unnecessary bugs into the next stage. He would also do a system update to make sure he had the latest versions of all the system code. After the piece of code passed the local test, a member would then submit it to the code server where it would be merged with other existing components.

The development team usually tried to adhere to the analysis and design that they created together at earlier stages. However, as the development work progressed, and the team gained a more in-depth understanding of the system, or as some new changes emerged in the technology or the industry, changes in the system would be necessary from time to

time. The team would hold group discussions about changes in features and designs to make collective decisions and create a common understanding of the latest requirements. As one member put it, they would never do anything without first informing the others.

2.2.7 Conflicts

Despite the fact that the company was built on a virtual basis and all of the members have never come together in one physical location, conflicts were relatively rare in their email communication. In the 9943 messages of the email archive (1997-1999), we coded only 277 messages, or 2.8%, as containing “disagreement” in category D. The period of greatest conflict was early in 1997, when the members were first learning to work together as a virtual team.⁷ Subsequently, levels of conflicts were even lower. Part of the reason, of course, might be that LC members were well aware of the term “flame war” and the limitations of email as a communication channel and tried to avoid conflicts in emails by moving these into direct phone calls or conference calls.

⁷ In the first four months of 1997, 117 messages (about 11% of total messages) are coded as “disagreement”, while the rate is much lower from then on – 160 “disagreement” messages (or under 2%) in a total of 8870 messages.

Chapter 3

RESEARCH FINDINGS

In Chapter 2, I discussed the members, context, and group behavior of the LC team using Olson & Olson (2001)'s framework. In this Chapter, I will focus on the coordination practices in LC and discuss my research findings. First, I will see how LC members coordinate their development activities in their distributed situation. Like other software teams, LC members needed to coordinate their activities to develop the LC system. In distributed situation, they usually needed to use somewhat different approaches to fulfil these coordination functions. Then, I compared the use of different coordination mechanisms in LC. Specifically, I will discuss the coordination tools available for LC members and how these tools combined to serve LC's coordination needs. Finally, I will study how their coordination patterns evolved over time.

3.1 Coordination Functions

In coordinating their software engineering activities, LC shared many challenges with co-located software teams. For example, they needed to fulfill a set of coordination functions in order to handle the various interdependencies among the team members. However, due to the geographical and technological limitations, LC members had to be adaptive and creative in carrying out these coordination functions. In this section I will focus on the following coordination functions: monitoring, tracking and control, notifications,

sequencing and synchronization, and task allocations (Malone & Crowston 1990; Pressman 1997).

3.1.1 Monitoring

Monitoring other member's activities/status provides activity awareness among team members. Due to the interdependencies among the collaborating team members, each member needs awareness of other members' activity status to carry out his/her own responsibilities within the team. And such activity awareness provides a way to coordinate the development activities of a team (Dourish & Bellotti, 1992). As Gutwin et al. (1996) pointed out, "workspace awareness reduces the effort needed to coordinate tasks and resources, helps people move between individual and shared activities, provides a context in which to interpret utterances, and allows anticipation of others' actions."

For distributed software teams, since their members' ability to directly observe what each other are doing with their work is very limited, people have to find ways to maintain awareness of the day-to-day project related activities of team members. No longer able to easily provide updates and status reports to each other through informal interpersonal interactions, distributed team members have to rely on explicit information exchange to gain awareness of team members, or to inform others about their own activities. "At best, these additional information transactions add to the costs of coordination, both in terms of time and effort, as well as in the potential for cognitive overload... At worst, the lack of information about others' activities can actually harm group morale, such as when team members assume their colleagues are inactive when they have not heard from them." (Steinfeld et al., 1999)

One important function of the weekly conference call among all members, for example, was for each member to report his recent activities as well as his current work plans. The presence of all LC members in the conference calls was important, not only in cases when there were group decisions to make and each member's input or vote was needed, but also in the sense that each member needed to report his status to the team and to learn about the status of the others. This enabled other members to monitor his progress.

Another important way to create awareness of team activities was using the team email list. LC members used the team email list extensively. They often included all team members in the “CC” list of their emails when discussing a specific technical event or problem about the LC system, even though the discussed topic might not fall in other team members’ working domains. As a result, such emails did not just exchange information between specific parties; they also created the opportunity of “overhearing” when all members were geographically dispersed. Team members could gain a basic sense of the issue being addressed and some understanding of other current work, without actively getting involved in the discussion themselves. In some instances, email was also used for formal status reports. When a member anticipated he would have to be absent from the weekly conference call, he would typically send out an email to the team email list to express his opinions on the issues to be discuss in the meeting and/or to report his work status. In the following email to the whole team, Keith reported his work status. At the same time, he also informed the team that he would not attend the conference call at regular time, and asked them to either reschedule the meeting or go ahead without him.

From: Fred
To: All
Subject: status and conference call tomorrow

Gentlemen -

I have been feeling more and more ill the last 3 days and I have not gotten much done. I am going to try to get to bed early :) tonight and not get up until late tomorrow in an attempt get over this bug. Consequently, I will not be able to take part in the conference call tomorrow morning. You can either go ahead without me or reschedule it to later, as you think best. I do expect to be in the office by noon.

regards..Fred

In LC, due to the flat team structure, monitoring was not just the responsibility of a project manager. Instead, all team members had many channels (such as the conference call or the team email list) through which to be aware of one another’s work progress and work plans. In other words, they were continuously monitoring one another’s work in an implicit and passive way. And such monitoring provided information for LC members to coordinate their activities accordingly.

3.1.2 Tracking and control

The effective sharing of artifacts among collaborators has been an important topic in the literature of computer-supported collaborative work. In distributed software development, while team members are working on different components separately, individual developers need to keep track of the progress and changes in other parts of the system to make sure the interdependencies among different parts work right. Software configuration management (SCM) refers to software engineering processes that enable software engineers to keep evolving software systems under control, and thus ensure that they will satisfy quality and schedule constraints (Estublier, 2000). In LC, where the development work was done by dispersed members, effective tracking and controlling of each member's development activities was even more crucial to the success of the project.

Two main aspects of tracking and control in the development of the LC system were bug tracking and change management. Bug tracking provided a process for identifying symptoms and sources of the bug and for searching for a solution to the problem, while change management provided control over the development of different components.

Bug Tracking: In traditional software engineering practice, a formal bug-tracking mechanism usually includes detailed information of each bug-report, status update, and bug-fix. In contrast, LC didn't employ a formal bug tracking mechanism in their practice. Because of their component-based development method, LC members implemented a component and then did a full component-wise test. Before the code was committed onto the server, any bug's potential ill effects were limited to the local system; and email messages did not contain much discussion of such bugs, except for cases when the cause of the problem had implications for other members' work, or when their help was needed to handle the problem. In the following example, Robert had just asked the team for help about a new problem he had just discovered. Keith replied to the email, suggesting possible causes; in addition he tried to track a previous problem with the "make" function that Robert had encountered earlier:

...

btw- what was the source of your other make problem, the one that you thought was a protection problem? If you got this far, you have obviously fixed it.

Keith

In his response, Robert reported the status of that earlier problem:

I have not fixed it. I have to execute the commands by hand.
I've reinstalled cygnus, I've tried to debug the make source.
I'm getting down to the option of reinstalling NT, which is the last thing I
want to do.

--
Robert

In this example, Robert had previously discussed his make problem with the team, but hadn't reported his status with the problem since then. When Keith wanted to track the latest progress of the problem, he had to explicitly ask Robert about it. So even though some bugs were reported to the team and discussed in the emails, the discussion itself didn't necessarily constitute a full bug-tracking record.

For bugs that emerged during integration testing of different members' work, LC members would usually inform the team, and then a group discussion via email and/or a discussion by telephone followed. Different solutions were tried and progress was reported until the problem was solved. In the following example, Dan encountered a crash during the system testing and sent out an email to the team:

From: Dan
To: All
Subject: an especially nasty bug.

I figured out was causing the crash. This is one of the better bugs
that I have encountered.

...
[description of the bug]

...
I'm going to see about fixing this in my area, CAREFULLY.
I've done major damage to the garbage collector, attempting
to find this bug, so it will be a while before I merge and
try to put a fix back.

Dan

Robert replied:

I plead innocent. Fred promised me that all allocations were in word multiples.

--
robert
robert@LC.com

Robert then tried to fix the bug in his part. After submitting a fix to the server, Robert sent out the following email addressing the bug issue:

From: Robert
To: All
Subject: inforequested bug

Keith:

I've checked in a new IntCode.Refine that should fix the problem, but I'll have to let you do the testing. If this does not fix things, you'll have to send me enough stuff to build your failing example.

--
robert
robert@LC.com

As we can see from the example above, though the email discussion captured some information about the bug-fixing process, LC members lacked an official bug-tracking mechanism with complete details of all bugs. But since all the email discussions were archived, LC members could still retrieve certain important information about the bugs from the email if necessary. So email archives served as an informal bug-tracking mechanism in LC.

Change Management: By using a central repository (the CVS server) as the change management tool, LC made it possible to control the integrity of a product that was developed by a distributed group of developers.

The CVS system works in this way: for each change made to a part of the product system, a team member has to develop it on a local copy of the component, test it against local copies of other parts of the system, and find and fix the problems, all before he checks in the new code to the server. All development work done in the private work place has to

be checked in to the public server before it becomes officially accepted. Before that, any changes made to any part of the system are limited to the local copies of the system. To reduce the risk of damage caused by problematic changes made to the system, the CVS server provided a “reverse” function that allowed users to “undo” changes and “reverse” back to a previous version of the system (Fogel, 2001).

LC members were cautious in their collaborative development efforts, avoiding changes to their part of the system without first making them known to the others and receiving no objections. As one outside observer (who served as the CEO of LC for a few months in 2001) put it:

“...They don’t do anything without the other person knowing it. Every email that was sent to the company goes to everyone in the company...”

Major changes that affected more than one part of the system were always discussed and agreed on before they were actually implemented. Small changes were also often communicated to other members when the author was not sure how it would affect other parts of the system.

By using emails and the CVS system for bug-tracking and change management purposes, LC members carried out the tracking and control function within LC adequately to develop the LC system.

3.1.3 Notification

Notification has been deemed to have important effects on team coordination (Shen & Sun, 2001; McCrickard et al., 2003). In a distributed work environment, team members have limited ways to keep track of the status of other members’ activities or the latest events happening to the team. Constantly actively monitoring other member’s activities or tracking the system’s status might involve too much overhead time and cost, and thus might become prohibitive for team members to do on a day-to-day basis. Under many situations, some researchers argued that event-driven notification is an important and effective way to provide activity awareness and status awareness within the team (Carroll et al., 2003).

Event-driven notifications send out an alert to a software development group about any pre-defined events, such as bugs, system updates, or incoming emails. Since it is event-driven, team members don't have to repeatedly spend time checking what is happening out there, when most of the time there is actually nothing happening. And it provides timely awareness of events, while regular active monitoring or tracking tend to have a delay (averaging half the checking interval) in detecting system or team events.

In LC, email served as the channel for sending out two types of such notifications: bug/defect notifications and "New on Server" Notifications.

Bug/defect notifications were issued when a developer observed a bug that might directly or indirectly affect more than one developer's work. Bug notification was sent out to alert other members about possible ill effects caused by the bug, to request help from other team members, or simply to notify the team about unexpected behaviors of the product. In the following example, Martin sent out a notification email to the team and asked for help:

From: Martin

To: All

Subject: io problem

There's an elusive bug that manifests itself by not flushing the last buffer in a random file (but repeatable on a machine if you don't change parameters). In my case it manifests itself during [...] by the following message:

[error message]

Keith encountered it in a different situation with a different file. I cannot reproduce Keith's situation on my machine.

However, the problem seems to be random -- if I add code to write additional information about file opening and closing everything is normal. If you have any advice on this please let me know.

Martin

During the development of the LC system, the product underwent many changes, each of which had to be transferred from a private work place to the public repository by a "check-in." Though the CVS server maintained a log of every check-in operation for each single file under management, it was not a common practice for LC members to maintain a regular updates-check on the server. Since the development work was done by

individual developers on a largely autonomous basis, it was hard for LC members to estimate when there might be an update on the server without explicit notifications from the author. “New on server” messages ensured that every member was aware of the recent changes made to the product and avoided potential conflicts and reworks.⁸

In addition, the archives of “New on server” notification emails on individual LC member’s machine served as records of the development activities. LC members could easily look at these records in situations such as identifying the possible cause of a bug. Dan described this function of the notification emails:

“..There’s this whole big folder of the new on server messages.... it was really important for purposes of just being able to figure out when something happened. Because you need to know, when you go to make a release, you need to know what you did, when you discover a bug, you need to know when you discovered the bug... “

Because we don’t have the record of the dyadic phone calls between LC members, I don’t know if they used phone calls for notification purpose. However, it is a reasonable assumption that they sometimes did use phone calls, probably just for notifying one another (instead of the whole team) about events that did not affect the whole team.

In general, LC members supplemented their monitoring and tracking/control with explicit, somewhat formalized notification.

3.1.4 Task allocation

Any complex system requiring more than two people’s efforts to develop involves task decomposition and allocation (Van de Ven, et al., 1976). The allocation of tasks among all the parties involved has important implications for the quality and cost of the finished product. Therefore, task allocation was an important issue in LC’s coordination of development work.⁹

Generally, to ensure the effectiveness and efficiency of the development process, the following issues need to be taken into considerations (Pressman, 1997):

⁸ Im et al. (2004) discuss the formats and the evolution of “new on server” notification emails in LC.

⁹ For a different approach to work distribution in LC, see Xu, 2004.

Mapping expertise/experience with development work: Though four of the LC members had Ph.D. degrees in Computer Science and all had rich knowledge of and good skills in the general technologies used in the LC system, their experiences and expertise differed somewhat in specific areas. Moreover, LC members were able to identify each member's strength and tried to map it to the development work (Xu, 2004). For example, Robert was deemed to have more experience dealing with the requirements analysis and the front-end of the system, Dan had direct work experience in the back end of the system, while Keith was more experienced in the so-called "middle end." As Robert put it:

“...this would correspond a lot with the way the compiler, the product, is organized because I've worked mostly on front end issues, reading the ... code and building. And Fred has done more of the kind of what's called the middle end issues of doing the optimizations of the [the LC system]. And Dan has done most of the back end issues...”

In an interview, Dan described the situation as follows (Xu, 2004):

“There is a round of who had to work on what. Because I had the most experience with [back end], I ended up working on it. So I could have worked on some other things. But no one else had worked on [back end], I ended up working on [back end]. Keith had more experience with attribute grammar. I felt that that was the way to work the [middle end]. So he ended up doing that. And I think Robert ended up writing [front end]. That was a case of specifications [which Robert was very good at].”

Mapping this experience to the development work took advantage of the relative strength of each member's expertise by assigning each task to the person who could do it most efficiently. Another advantage of mapping the tasks to expertise was that it reduced coordination costs.

As the development process went on, problems emerged and unexpected tasks were identified. Since each member's expertise was known to the team, the assignment of new tasks was often obvious, really simplifying the dynamic task allocation in later stages of the development, when LC members had to spend a lot of time dealing with bugs:

“...we all know our expertise. We know who is the real expert on any kind of a thing, and we've gotten good enough so that we can kind of smell a bug... it's Dan's, Robert's or my bug and we get it right most of the time....” (From interview with Keith)

Although this mapping was not always possible, LC members clearly used it when they could.

Workload balancing among team members: Balancing workload among members improved the effectiveness of the team. LC members seemed to understand the importance of workload balance and tried to maintain such balance. When LC members discussed the task allocation at the very beginning, workload was roughly estimated and balanced out among members. For example, though Dan was the expert and had more experiences in so-called “garbage collection” than any other LC member did, he had already taken up a lot of other assignments. So Robert took up this area. And since LC members reported their work status (mainly through weekly conference calls), they generally had a good understanding of each other’s work load. When there were things that were just waiting to be done, those with the least current work did them. As Robert described it:

“There were a couple things that were just kind of there, and somebody had to do them. And the person who had the least other stuff to do did them.”

Thus, expertise was not the only criterion used in assigning work.

Reducing interfaces and interdependencies between sites: The system’s architecture determines the interdependencies among different parts, and determines whether these parts can be allocated to different development sites without causing serious coordination problems. It is common practice to design products according to the principles of modularity: software components’ structures should be well defined and have few interdependencies with one another. This practice reduces the complexity of systems and lowers the coordination cost.¹⁰ The modular components of the product may then be assigned to each separate development site to be finished on a relatively autonomous basis, before they are merged into a single product at the end.

¹⁰ In a study conducted by Leonard and colleagues in the large US-based consulting firm AMS, the virtual teams did have lower interdependence than did co-located teams. For more information, see Leonard et al., 1997.

In LC, dividing the LC system into three main modules and assigning them to different LC members helped to reduce the interfaces and interdependencies between different sites. An LC member¹¹ talked about this in an interview:

INTERVIEWEE: ...It's easier to ... segregate to a big extent.

INTERVIEWER: OK, so you're with the divide and conquer kind of approach.

INTERVIEWEE: ... And it really is... the best way to do it. And if you can divide things now nicely, it just works better that way and [...] that's what we do, implicitly.

Despite the modular method, as the situation changed (especially in later stages of the LC system development), each LC member's specific tasks also changed. And the modularity might not be exactly maintained, especially in smaller tasks. A list of the LC system's components and the corresponding developers (provided by Dan) shows that roughly half of the components were done by two or more developers. Such a situation unavoidably affected the interdependencies between the LC members (and their tasks).

3.1.5 Sequencing & synchronization¹²

The need for sequencing two or more developers' activities arises when these activities have sequential dependencies, that is, when several pieces of work needed to be done in a specific order to get the proper results. For example, whenever producer-consumer interdependencies exist, the producer needs to produce something that is required for the consumer to start the next step of work.

As we could see from the previous section, due to reasons such as the inherent interdependencies among system modules and the dynamic allocation/reallocation of tasks among LC members, the activities within the software development team needed to occur at an appropriate pace with relation to each other, even though distributed software developers usually enjoy much more freedom and flexibility in their schedule than they did in traditional work environments. As Robert said in his interview,

¹¹ This LC member could not be identified from the record of the interview, at which all three full-time LC members were present.

¹² For more discussion on the sequencing and synchronization in LC's practice, see Im, Yates, & Orlikowski, 2004

“But it was clear that I was going to keep track of all of the dependencies between the [WHATEVER] classes and things that we were compiling in order to make sure that when you compiled a program that everything was correct with respect to, with everything compiled in the right order.”

While event-based sequencing deals with proper working order, synchronization refers to the fact that LC members needed to arrange their development work so that certain tasks would be accomplished before a given time. Such given time could be scheduled public release time, scheduled internal system integration test time, or any other informal timelines.

In the following example, Robert proposed to the team some new changes in the coding syntax, generating team discussion. If the new syntax was accepted (and it seemed that it would be), it would affect an update that Keith had planned to submit later that day. Keith didn't want to delay his update and interrupt the work schedule, nor did he want to cause any problem by committing an update that was not compatible with the new syntax. So he sent Robert an email informing him about the update due in a few hours:

Date: 1997-04-24T18:44:12 GMT
From: Keith
To: Robert
Subject: Re: three proposals

....

I am a couple of hours away from putting an update out and I want to know what to do?

Keith

Robert replied with the following message:

Date: 1997-04-24T19:24:06 GMT
From: Robert
To: Keith
Subject: Re: three proposals

> I am a couple of hours away from putting an update out and I want to
> know what to do?
I'll try to have the parser and the documentation updated sometime this evening.

--
Robert
Robert@LC.com

Two hours later, Robert completed the necessary update of the system so that it would accept the new syntax. He sent an email back to Keith again, informing him that the new changes were in place, and he could commit his planned update to the system:

Date: 1997-04-24T21:30:23 GMT
From: Robert
To: Keith
Subject: Re: three proposals

> I am a couple of hours away from putting an update out and I want to
> know what to do?
I just committed the Link package, Link.txt, and ObjectFormat.rtf
The smarts are not there, but it seems to accept the new syntax.

--
Robert
Robert@LC.com

In the above example, Keith and Robert sequenced their work through synchronization with Keith's informal timeline. Keith informed Robert about the coming update in a few hours, setting up a common time reference (synchronization), and (implicitly) requesting Robert to sequence their work by getting the new syntax in place at the time Keith had scheduled to submit the new update.

In a distributed work environment, it is more difficult than in a co-located workplace to foresee what would happen at each distributed site, where each team member faced a totally different environment and situation. For example, Dan was working at home and sometimes would do housekeeping activities during day time; Robert had a habit of bicycling; and Keith would go out for a walk from time to time during work hours. And things got even more complicated because they worked in different time zones, and one person's work time overlapped with another's off time. Facing many unforeseeable disruptions to each member's work schedule, frequent scheduling and rescheduling was common in LC's email communications as well as in conference calls to sequence and

synchronize their work. During the three-year period of study (1997-1999), among a total of 9943 messages, we observe 175 (averaging 1.12 per week) sequencing messages and 775 (averaging 4.95 per week) synchronizing messages. The difference may reflect the fact that sequencing emails were used when LC members' work was closely interdependent (which was not so common due to the module-based development methodology in LC), while synchronizing emails were used when LC members (such as product releases or fixes that were to be submitted into the CVS server) had a common timeline so they had to frequently synchronize their schedule to avoid conflicts.

In general, LC members usually were adaptive in carrying out their coordination functions. As we have seen from the above discussions, they often used a combination of two or more coordination mechanisms in carrying out coordination functions. For example, they used both phone calls and emails for sequencing and synchronizing their activities. This practice surely helped LC members' coordination in a distributed situation.

3.2 Coordination Mechanisms

Various mechanisms (tools to fulfill coordination functions) are available to allow co-located software development teams to meet their coordination needs. However, for distributed software teams such as LC, the options of coordination mechanisms were limited by the physical separation of members, eliminating rich face-to-face interaction channels among them.

The limitations on available financial, human, and technological resources made LC members highly selective in choosing coordination mechanisms in their distributed development work. Cost, reliability, and usability were the main factors they considered in doing so.

In LC's practice, the team email list, conference calls, dyadic emails, dyadic phone calls, the CVS system, norms, and documentation were the main mechanisms employed in coordinating the project's progress and the team's activities. In the following sections, I will discuss the role of each mechanism in the coordination of development activities in LC.

3.2.1 Team email list

Since almost all of the hardware and software were readily available, and since most email client software could run in parallel to the software development tools, email was one of two standard communication media used by LC from the very beginning of the project. In addition, email provided great communication flexibility for LC members working in their home offices.

The frequently used LC team email list played an important role in LC's coordination. The team email list was not formally established until LC obtained email server services from an Internet services company. Before that, LC member simply included each member's email address in the recipient list when he wanted to send an email to the whole team.

The team email list served as a public broadcast system in LC. Instant delivery of messages to all recipients' email boxes made the team email list an ideal option for event notifications. Even though there was no guarantee that each recipient was at his computer and ready to read the message as soon as it arrived, the sender could make a reasonable assumption that other members would see such a notification when they were actually doing their development work.

The use of the team email list as the notification channel within LC was flexible. It could be used for alerts about strange behavior of the product under development, or it could be used to inform the team about recent changes committed onto the server. Whatever the specific purpose was, the notifications through email provided a way for all team members to maintain a "common ground" about the status of the team as well as the progress of the project.

The team email list also played a crucial role in allowing team members to monitor each other's activities and progress and providing group awareness within LC. In LC, it was common practice that when two members were discussing technical issues about their work, they often made the discussion transparent to the team by sending a copy of the message to the team email list, even when the issue did not directly affect other members' work or the system as a whole (see Table 3.1).

Sender	Emails with all members in "To" or "CC" list	Total emails sent out	Percentage
Dan	2625	3634	72.23%
Keith	1977	3641	54.30%
Robert	2423	3331	72.74%
Fred	243	346	70.23%
Martin	465	586	79.35%

Table 3.1 Percentage of emails with the whole team as recipient (from 1997 to 1999)¹³

As we observed from the above table, all members except Keith sent out over seventy percent of their emails to the whole team, and even Keith sent out over half to the team.¹⁴

Regardless of the specific intentions, in effect the practice of using the team email list did allow team members to be aware of each other's current work and thus to create the opportunity of collective thinking and to avoid duplicated efforts. It also provided the opportunity to make better use of team members' expertise. However, such use of the team email list could also cause some problems in the practice. For example, since it was difficult to determine what kinds of discussion were beneficial to the whole team, LC members usually just selected to send most emails to the team list. One risk was that less relevant discussions might overwhelm each member's email box. Fortunately, LC was a small team with only five members, of which two were part-timers and less involved in the daily activities. Consequently, the average team emails per day were still at an acceptable range (8.74 per day from Jan 1997 to Dec 1999).

In a personal communication with the author (July 31, 2004), Dan gave several explanations for this phenomenon, supporting the above reasoning:

- a) Sometimes an "outsider" [an LC member that was not directly related to the specific problem] would see something or think of something that the other two had missed, or perhaps he had seen the problem in some other context.
- b) ... there was sort of a norm that everyone should always know what was going on.

¹³ The numbers in this table might be skewed slightly by the fact that we don't have all the dyadic emails; in particular, we are missing those between Fred and Martin, as well as some of those between those two part-time members and Keith.

¹⁴ For more discussion, please see section 3.2.2. "Dyadic emails."

- c) The cost of doing this is relatively low -- consider the volume of spam we get nowadays, consider the volume of essentially useless email that is set within any large organization.

Sometimes the team email list was also used for team members' status reports. It was very common that LC members used emails to explicitly report their work status to the team so that other members working on relevant parts could adjust their activities, as in the following example:

Date: 1997-05-07T18:47:06 GMT
FROM:
TO: ALL

I got SOMETHING to link with our version of SOMEFILE-A and SOMEFILE-B.

I have not tried Dan's new makefiles.

However, I put a new copy of the Tests/Makefile out that does the proper thing for linking SOMETHING against our library.

I also put a new version of SOMEFILE-C.
This one has the [...] relocation commented out. There were other bugs fixed so get everything and start from scratch.

Robert can look for the [...] comment to uncomment this block if he wants a test bed.

However, I believe that we are in position to test method calls when Dan is ready with the code.

Keith

In the above email, Keith mentioned the progress in his work ("I got SOMETHING to link with our version of SOMEFILE-A and SOMEFILE-B."), as well as the status of a task related to Dan ("I have not tried Dan's new makefiles."). He also let Robert know what to do if he wanted to do a test. In addition, he talked about synchronizing the team's testing work ("I believe that we are in position to test method calls when Dan is ready with the code."). What is more, Fred and Martin also got an update on the work and knew what they could do with the new code; even though this email didn't explicitly tell them what they should do. Such coordination emails ensured that LC members maintained

good awareness of the progress in different areas of the project and could coordinate the team's activities accordingly.

The fact that the team email list was also frequently used for technical discussion such as bug-fixing made it an informal bug-tracking tool. Since every LC member received a copy of all the team messages, it was very convenient for them to look into the email discussions to track the status of bugs.

The team email list was also used for ad hoc scheduling. LC had a regular weekly conference call for all members, and LC members sometimes sent out an email to the team email list to confirm the meeting in advance and reschedule the meeting if needed.

LC members were reluctant to use the team email list for task allocation purposes. One reason was that the negotiation process usually involved a lot of interactions in a short period. As Dan stated in an interview: "I don't think that we ever did any negotiation over email, whereas you can do that on the phone." Another reason was that, in contrast to simple technical discussions, task allocation frequently involved personal relationships or conflicts of interest in addition to knowledge and skills. In such situations, face-to-face meetings or conference calls served better for the task allocation purpose. Because LC never met as a team, much of the task allocation or re-allocation was handled during the weekly conference calls, while at the same time email exchanges also played an important role in dynamic task allocation.¹⁵

In summary, the team list was used for the notification, monitoring, tracking and control, and sequencing and synchronization coordination functions in LC's practice.

3.2.2 Dyadic emails¹⁶

Besides the team email list, dyadic emails also helped LC members in coordinating their activities. Sometimes LC members would use dyadic emails to coordinate their work, especially when they were working together to address an issue, or when a task was being transferred from one member to the other. In the following example, Keith encountered a

¹⁵ Xu (2004) discusses the explicit and implicit forms of work distribution in LC.

¹⁶ For more discussion on dyadic emails, also see Ghosh, Yates, & Orlikowski, 2004.

problem that he believed was directly related to Dan's work and less relevant to other LC members' work, so he notified Dan about this problem without sending a copy to the whole team. In addition, he also told Dan about his current status and work plan:

From: Keith
To: Dan
Subject: error

When did you produce this list?
It was my hope that the update that I put on the server at around 6:30
would have fixed some if not all of the first group.

...
[error info]

I am still working on the second group.

.....

I will deal with these tomorrow. These are caused by floating point errors inside a speculated region. I need to figure out the proper semantics then I can hack a solution.

....
Keith

However, such instances of dyadic coordination emails were very rare. About 95% of the dyadic emails between LC members were discussions of technical work without explicit coordination. It seems that when LC members coordinate their work with others', they preferred to do so with the whole team instead of dyadically. Such practice is reasonable since it reduced coordination cost and facilitated team awareness: LC members didn't need to coordinate with one another separately, and by coordinating in team list emails, they avoided possible errors resulting from overwhelming dyadic coordination.

Though there were not many explicit coordination emails between two LC members, it seems that these dyadic emails provided valuable information about LC members' specific current work. Such information allowed LC members to be aware of each other's work and (implicitly) monitor each other's work progress.

Sender \ Receiver	Dan (only)	Keith (only)	Robert (only)	Fred (only)	Martin (only)	ALL	Other
Dan		16.76%	5.04%	1.21%	0.69%	71.08%	5.23%
Keith	19.83%		28.43%	0.58%	1.51%	42.02%	7.64%
Robert	12.10%	25.88%		2.25%	2.49%	56.08%	1.20%
Fred	13.29%	4.34%	16.76%		1.16%	56.07%	8.38%
Martin	6.66%	3.58%	7.51%	1.19%		73.04%	8.02%

Table 3.2 Dyadic emails between LC members (from 1997-1999)¹⁷

From Table 3.2, we see different patterns among the three full time members. For example, among the three full time members, Dan had over 3 times as many dyadic emails to Keith as to Robert (17% vs. 5%); Keith had only slightly more dyadic emails to Robert (28%) than to Dan (20%); and Robert had over twice as many to Keith (26%) as to Dan (12%). In general, this corresponds quite well to the actual work relations among the three members, as stated by Robert:

“...you could draw a little diagram with Keith in the middle and me on one side and Dan on the one side. So, Keith did have lots of interactions with Dan, and I had lots of interactions with Keith. And Dan and I haven’t really, we’ve worked on some stuff but, in comparison, we haven’t had as much stuff that we worked on together.”

We should notice, however, that due to the different personal preferences over communication channels such as emails and phone calls, the numbers here might not reflect the exact communication intensities among specific pairs of LC members. For example, Keith and Dan had twice as many dyadic phone conversations as Keith had with Robert (100 calls/month vs. 41 calls/month)¹⁸. Nonetheless, we could reasonably assume that the email communication patterns generally reflected the relations between the LC members. Monitoring may thus have been the most important coordination function of these dyadic emails, with notification a distant second.

In general, dyadic email discussions between LC members provided the opportunity for them to implicitly monitor each other’s work progress. In limited cases, LC members also used dyadic emails to sequence and synchronize their work.

¹⁷ Again, since we don’t have all the dyadic emails, the numbers here might be low, especially for dyadic emails including Fred and Martin.

¹⁸ From a personal communication to Tanu Ghosh (July 29, 2004).

3.2.3 Conference calls

Besides emails, the conference call was another major communication channel and coordination tool within LC. Comparable to the review meetings in traditional co-located software teams, weekly conference calls played an important role in the project control of distributed software teams. During the conference calls, topics such as business strategy, administrative issues, and technology strategy were covered to ensure the operations of the company as well as the progress of the product development. In this research, I focus on the coordination functions of conference calls.

One of the most important functions of conference calls was that they provided tracking and control in the distributed software team. During the conference calls, LC members went over a set of topics to assess recent project progress, identify deviations from the original plan and schedule, and collectively find a way to take corrective actions. Serving as a mechanism for open discussions of a lot of issues, conference calls allowed LC members to understand plans and progress of the project, develop common understandings of current and potential problems, and ensure commitments from the participants.

The minute for a conference call on August 19, 1998¹⁹ will serve as an example of the functions of such conferences:

LC	19 August 1998	Everyone present
Status		
Keith		
	- wrote classes we need to get computer	
	o running may have a few dependencies into bad [COMPANY-A] stuff that needs to be	
	found	
Dan		

¹⁹ The original conference call minutes were hand-written by an LC member, and were later transcribed. Problems such as indecipherable handwriting have made perfect transcription impossible; however we did try to keep all the information in the minutes. As always, the name of companies and people, as well as other sensitive information, were disguised for the purpose of confidentiality.

- currency library... talking to [COMPANY-B]
- internationalization
- vacation Tues. 25-7, Tuesday 1 Sept.

Robert

- linear
- libraries almost done

Fred

- [SOMEBODYA] at [COMPANYB] – [SOMEBODYA @ COMPANY-C.com]
- fp white pages, application mailing list
- fixed best _____ forwarding
- contacted lawyers about draft license agreement

Martin

- ran out of disk space
- will get disk from Fred
- will follow up on [COMPANY-B] libraries

beta release

- September 16 date
- [SOMECOUNTRY]
- [SOMEBODY's company]

Things to fix

- exploration of code - use [MSEC/insec?] - Fred will supply code, Robert will inset
- Properties
- o properties set by _____ [WHATEVER-A] < Robert
- o WHATEVER-B <
- o OpenGL, etc.- Keith will define, look at registry
- Better defaults
 - Make default, Dan corrects, Robert _____ incremental compile/incremental link - two commands
- Installation Program, Fred except noted
 - copies things to the right place, Keith will start a list
 - builds libraries from 1.2 beta4 >/Robert machinery >, Keith spec
 - display license before installation

examples:

```
helloworld      }
benchmark} Martin
use DOS files   }
```

```
Testing                }
target date 9th September } Martin
```

User manual _____

Robert (after defaults) Dan
Then onto

In this conference call, several coordination functions were carried out:

1. **Monitoring:** Each LC member reported his activities and accomplishments since the last meeting, as well as his current task and plans for next steps. These status reports were a crucial factor that maintained team awareness among the members and allowing LC members to monitor each other's work progress, ensuring effective coordination among distributed developers. Besides allowing all members to better understand one another's task and status, this weekly status report also created the opportunity for them to check their own progress, and see how well they had been doing in comparison with other members. LC members also used this opportunity to report their problems to the team and seek solutions. In this example, Martin reported his storage space shortage, and the solution was to get a new disk from Fred.
2. **Tracking and control:** during conference calls, LC members sometimes talked about problems/bugs they found in development work. This was usually a continuation of email discussions previously held among the members, in which they decided that the problem/bug was an important one that could affect the whole project and should be brought into all members' attentions. In this example, LC members talked about several "things to fix".
3. **Task allocation:** Conference calls were also where LC members allocated most of the tasks. Besides the original task allocation at the beginning of the project, LC members also needed to decide the allocation of tasks as a result of newly identified problems or reworks as the project moved on. In the example above, LC members discussed several technical problems and specified each member's responsibilities, reducing the uncertainty in work arrangement ("exploration of code ... Fred will supply code, Robert will inset").
4. **Synchronization and sequencing:** Conference calls were also used for scheduling and coordinating LC members' activities. Due to the interdependencies among LC members and among their tasks, LC members often worked closely and

maintained frequent communications in the team. However, when some member needed to take off for some amount of time (for business or other reasons) during which he might be absent from the communications as well as from the development activities, it might disrupt other member's work. To avoid such disruptions, LC members usually announced their expected absence during the conference calls, so that those affected members could find a way to continue their work. In the example above, Dan announced his forthcoming vacation between Aug 25-27 and Sept 1. In addition, LC members also set a target testing date of September 9 and a public release date of September 16 so that all members could arrange their work accordingly. In this conference call, LC members also arranged a work flow to finish the user manual: from Robert to Dan and then onto Martin.

It is noteworthy that LC's weekly conference calls were different from the traditional project reviews in a general sense. In traditional software engineering, project reviews, technical and quality reviews, and management review meetings were all important meetings and were complementary to one another in ensuring the project's progress. Project reviews in traditional software engineering mainly served as coordination and control mechanisms, and technical and quality reviews are designed to detect and correct technical and quality issues, while management review meetings are held by a senior management team or committee (Jurison, 1999). In LC, conference calls were often a combination of project reviews and management review meetings (at one point in the first year, they separated out technical issues into different meetings than business issues).

Due to the multiple coordination functions of the weekly conference calls, the presence of all LC members was very important. These conference calls were scheduled at a fixed time every week, deliberately selected so that it accommodated members' schedules in different time zones.

Sometimes LC members also used three-way calling for discussions in a similar way they used the conference calls.

3.2.4 Dyadic phone calls²⁰

Besides regular weekly conference calls that served the needs of formal meetings in LC, dyadic phone calls were extensively used in LC's practices. LC members often had ten or more dyadic phone calls in a normal workday (Ghosh et al., 2004). For example, on a specific day, Dan had 18 dyadic phone calls (lasting from one minute to thirteen minutes) with other LC members, in addition to a conference call with all other LC members (see Appendix B).

Though we are not able to know the exact content of the conversations over phone, from our interviews with LC members, we do know that a lot of coordination was going on through phone calls:

INTERVIEWER: So, that's where you do a lot of the coordinating work and the sort of who does what on the phone?

Dan: Yeah, I think so. I think so, because you can't really tell on email whether, I mean, you can give people information. But I don't think that we ever did any negotiation over email, whereas you can do that on the phone.

Though Dan was referring primarily to the use of conference calls, we can reasonably assume that similar coordination functions -- task allocation and sequencing/synchronization -- were also carried out in dyadic phone calls.

Dyadic phone calls between two LC members served the needs for dynamic and informal coordination channels in LC. And in a certain sense, dyadic phone calls were probably used in place of the frequent informal interactions in co-located software teams.

3.2.5 CVS system

As we see in the previous section, to make sure that the work done by different developers can come together as a whole, and that the integrity of the source code is not compromised by multiple developers' activities, LC used CVS (Concurrent Versions System) as the platform for their collaborative development activities. CVS provides the ability for users to track (and potentially to reverse) incremental changes to files and can be used concurrently by many developers.

²⁰ Ghosh et al. (2004) discuss in detail about the patterns of using dyadic phone calls in LC.

In LC's case, CVS played a critical role in the successful development of the system. After developers agreed on the system design of the product and the interfaces between different components, they could go back and implement the components individually in their own home offices. Though these developers maintained close contact and held weekly conference calls, they had a lot of flexibility in determining how to do their implementation work. After components were developed, they would be submitted onto the central CVS server, where all code was merged into a single product system there.

After one member's code was submitted to the CVS server, other members could check out a copy of the code from the server to put on their local computers for testing purposes. Each member would test his work against a copy of other system components to check for bugs, incompatibilities, or inconsistencies.

As the change management system, CVS provided a way for LC members to track and have control of the latest changes made to the product if necessary. Every developer had to check his latest code into the CVS server after he finished a new piece, refined a chunk of code, or fixed a bug. LC members each agreed to send out a "New on server" notification email with a brief description of the new changes to the team email list (sometimes just the confirmation message from the CVS server). An example of such an email is as follows:

```
DATE: 1999-01-05T01:13:31 GMT
FROM: Keith
TO: ALL
SUBJECT: new code on server
```

```
fixed bug so that the world ended if you had an WHATEVER env variable.
CVS: -----
CVS: Enter Log. Lines beginning with `CVS: ' are removed automatically
CVS:
CVS: Committing in .
CVS:
CVS: Modified Files:
CVS:  .../.../.../ WHATEVERDIREC -a/WHATEVERFILE-a
CVS:  .../.../.../ WHATEVERDIREC -b/WHATEVERFILE-b
CVS: -----
```

Such changes would be available on the server to all the team members. Every user could check for the latest submits on the server by looking at the CVS log, and get a copy of the

latest changes by a CVS “update” operation. This provided a mechanism to keep all developers’ local copies of the code up-to-date and consistent with all other members’ copies.

CVS also provided the flexibility to “undo” changes already committed onto the server. Such reversion functionality was important at the early stages of software development, when there were many uncertainties and possible errors. Considering the communication barriers and coordination difficulties in distributed software teams, this capability also served as an important back-up measure for damage from coordination problems or miscommunications.

By looking at the CVS log, an LC member could monitor the update activities related to any single file on the server. If necessary, it was possible to monitor the progress of the project by simply looking at the update records (which included information about date/time, lines added/removed, and brief comments by the author) in the CVS log. CVS didn’t provide the author of the changes, but the three fulltime LC members should have been able to determine the source of problematic code even without additional documentation such as the “New on Server” emails, since each knew what piece the others were working on. An outsider or one of the part-time members would need the “New on server” emails to judge.

Another shortcoming of CVS system was that though it kept all the update records of every single file in the system, it didn’t maintain a record about what files were involved in a single check-in. Because changes to several files were in fact combined as one logical change, LC members might need to keep track of what files were affected by an update. If an LC member checked in several files in a single CVS operation, the CVS system would not keep a note that these files were checked in together. The only ways to figure out this information were either to look at the log messages (date/time, comments) of those files to decide if they were submitted together, or to rely on the “New on server” notification emails (which usually included a list of the files involved) sent out by the author.

In summary, CVS systems played an important role in the tracking and control of development activities in LC.

3.2.6 Norms²¹

Norms are another mechanism for coordination. Unger et al (1977) defined norms as follows: “the patterns or habits of interpersonal interaction that members use to accomplish [...] tasks define the communication norms of a team.”

LC members worked in a loosely structured and sometimes ad hoc manner. Though they used mechanisms such as conference calls and team email lists, most of the time they worked without explicit coordination. Thus, norms played an important role in regulating and coordinating each team member’s activities, especially in providing tracking and control in LC’s practice.

Team norms concern how team members interact with others and fulfill their responsibilities within a team. Due to the interdependence among LC members and their tasks and the geographic distribution of the team members, communication and development norms had a great impact on the effectiveness of the distributed team.

Communication norms: LC members worked in their own home offices, distributed in different time zones in the US, and they needed to rely on both synchronous communication tools (phone calls) as well as asynchronous ones (such as emails) to share information. The physically separated team members’ awareness of each others’ current activities was thus limited and usually not up-to-the-minute. This created communication barriers in LC members’ daily work and required careful consideration. LC members usually implicitly agreed on a certain way of using the multiple available communications channels: phone, email, and conference calls. A common understanding of how to use different communications channels allowed LC members to better communicate with one another and to avoid potential conflicts resulting from inappropriate use of communication channels: email was mainly used for asynchronous communications such as notices and technical discussions; for urgent problems,

²¹ For detailed discussion on the role of norms in coordinating LC members’ activities, please see Ghosh, Yates, & Orlikowski. 2004.

discussions on complicated issues (usually involving non-technical factors such as negotiations) or on issues considered inappropriate to be recorded in email archives, phone calls were a more appropriate communications channel; conference calls were used for official discussion of general issues (such as business, marketing, technical strategies, etc.) and non-urgent topics. Communicating with one another in accordance with these mutually accepted rules was one of the reasons that the distributed team operated relatively smoothly (Ghosh et al., 2004).

As shown in Ghosh et al. (2004), some norms developed over time during the first year of LC's existence. For example, they cite the accommodation that Robert made to Keith's preference for phones, in spite of Robert's preference for email, for many types of communication.

Development norms: Even though LC members tried to decompose their work and assign the tasks to each member in such a way that they were relatively free to do the development work on their local computers in their own house offices, they still needed to merge their codes on the central server and make sure that each component worked together without compatibility problems. For example, even if a component could run without problems on its own, it might not work at all after it was integrated into the whole system, either because of a faulty function in another component developed by another LC member, or because of incompatibilities between different coding styles.

Development norms served as the behavioral standard that every LC member was supposed to follow in order to avoid problems in team development activities. One example was that each member would try not to introduce bugs into the public storage space (i.e. the CVS server). As Dan pointed out in an interview, "...we tried to adopt a policy of testing ... before we put new code in. And so, we rarely, rarely introduced new bugs." Though LC members were largely free to experiment and work in their own styles when dealing with their work in their local work space, they agreed to take great caution in transferring their code onto the CVS server. To do this, they usually fully tested their code against local copies of other parts of the system, in order to find out all abnormal behaviors and fix them. And after they finished such testing and fixed all the bugs, they

would do a system update to see if there were any new change on the server, even when they hadn't receive any "New on server" notification. As a result of this practice, the code submitted to the server was usually fully-tested and largely bug-free, significantly reducing the quality risks in the distributed development and avoiding potential damage from bad coordination.

Such communication and development norms were sometimes the result of group discussions, but were often behavioral patterns that simply emerged in the daily activities and were later recognized and accepted by all LC members. One example is the way LC members used email and dyadic phone calls in their work. Originally, Robert and Keith had different preferences for using email and phone calls: Robert preferred to use email more for their daily communication, while Keith liked to pick up the phone and talk to another person directly. There were some conflicts among the LC members, but later they learned to find a way to accommodate different personal preferences and combine the use of phone calls and emails in different situations (Ghosh et al., 2004).

Both communication norms and development norms helped LC members to control their distributed work.

3.2.7 Documentation

Technical documentation plays an important role in today's software engineering. According to Thomas and Tilley (2001): "Software engineers rely on program documentation as an aid in understanding the functional nature, high-level design, and implementation details of complex applications".

In software teams, the software development process is accomplished by a group of people cooperating. Communication barriers and functional differences, among many other factors, make common understanding difficult to maintain. Shared technical documentation makes sure the team has a common view of the system and thus helps the software engineers to coordinate their interactions.

In LC's practice, documentation mainly provided tracking and control to the LC members. There were mainly two types of documentation that LC members used in their practice: design documentation and code documentation.

LC maintained a set of *design documentation* to provide guidance in structuring the system. The documentation was very important in that it clarified the architecture of the system as well as the interfaces among different LC members' work. However, LC members didn't enforce a formal mechanism in keeping and updating the documentation (a practice that caused some problems), possibly because they communicated frequently, especially during conference calls, and they could clarify things in the phone calls or email discussions:

“We did have some technical documentation that we checked in some times describing especially tricky parts, but I don't think it was kept up-to-date. We would try to, but it was one of these things when you after six months [say] “oh yeah, I'd better take care of that.” (From interview with Dan)

Code documentation refers to the comments written between lines of code to describe a specific implementation issue, such as a detailed structure of a class, or a rationale behind a specific method. Dan explained it in this way:

“The other place where you would get that kind of technical documentation is in the code itself and say you might sometimes find gigantic comments describing the approach to your particular problem.”

According to Dan, the advantage of putting documentations in the code, rather than writing it in a separate formal documentation file, was that it made using and maintaining the documentation easier:

“... if you are working on the code, there is the documentation. You don't have to go look for it and [you would] more likely [...] be reminded when you change the code in some way. If you have to change the behavior or requirements. ... you know if you forgot, the documentation is there.”

The code documentation did not just serve as a reminder for the author himself, but also served as a communication channel in facilitating the cooperation of two or more LC members on one component. As Dan noted:

Interviewer: So you actually write a lot of comments in your code?

Dan: In places - in other places, not much. I think it was, I think you would find more strategic comments than technical comments, you would also find if there is a place that someone is going to get confused and to un-confuse themselves they might leave a little bit of comments [so] that next time they come through, they wouldn't get re-confused... [the comments] are for anyone, any of us, to look at the codes.

The different coordination mechanisms in LC combined to carry out different coordination functions which were crucial to the successful development of the LC system (see Table 3.3 on next page). By adaptively using these coordination mechanisms, the distributed LC members together successfully developed the LC system, even though they were not so successful on the business side of the company.

Coordination Mechanisms	Corresponding coordination Functions
Team email list	Notifications. Monitoring Sequencing and synchronization Tracking and control
Dyadic emails	Monitoring (implicitly) Sequencing and synchronization (limited)
Conference calls	Monitoring Sequencing and synchronization Tracking and control Task allocation
Dyadic phone calls	Task allocation Sequencing and synchronization (possible)
CVS system	Tracking and control
Norms	Tracking and control
Documentation	Tracking and control

Table 3.3 The use of coordination mechanisms in LC

3.3 Dynamics in Software Engineering Coordination

Since coordination is about interdependencies among team members and tasks (Malone & Crowston, 1990), we can expect coordination needs in software engineering to change as the interdependencies in the team change. Better understanding of the dynamics in coordination needs throughout the software engineering process will help us increase the effectiveness of coordination mechanisms in software engineering practices.

After the first prototype of the LC system was developed in early 1998, LC members continued to revise and refine it for several reasons:

1. Bug-fixing. Fixing bugs identified in internal and external testing was a necessary step in software engineering in order to deliver a usable and reliable product.
2. Performance improvement: Benchmarking with competing products led to revising the product to gain better competitive position in performance.
3. Conversion to new technical changes: Changes in technology standards required corresponding changes in the product.

In this section, we will studying how the coordination activities in LC changed as their activities changed over time.

For development activities: Since all LC members submitted all their work onto the CVS server, we assume that file submission activities on the CVS server reflect the development activities during the same time period. Figure 3.1. shows the numers of files checked into the CVS server from 1998 to 1999. By studying this figure, we could have a basic concept of how LC members done their development work over time.

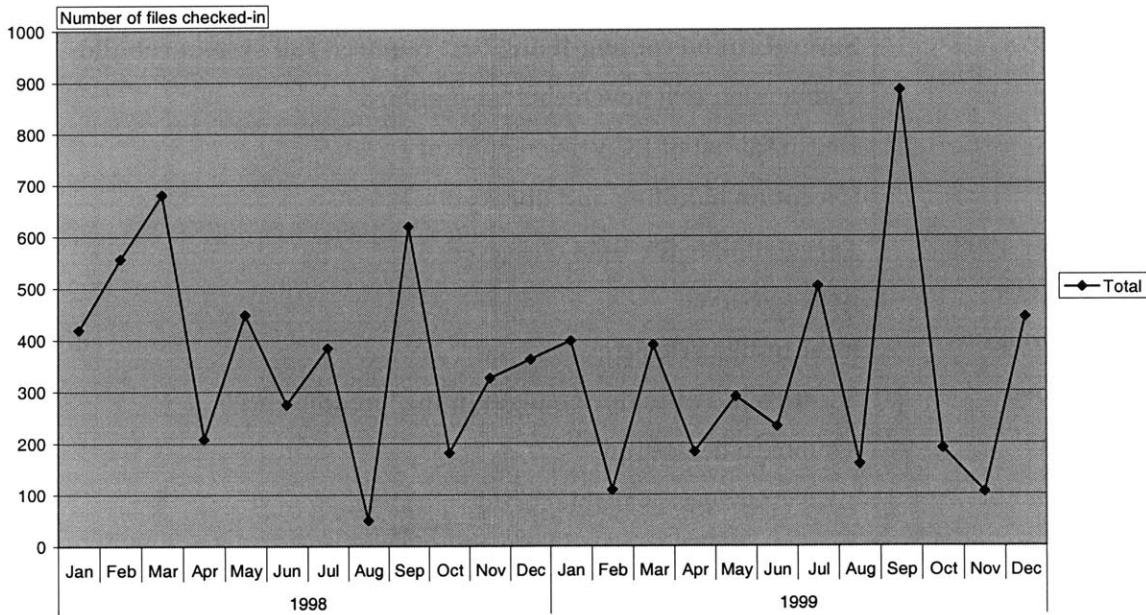


Figure 3.1. Files checked in to the CVS system (1998-1999)

For coordination activities: Since email was used as an important coordination channel, we use the email data to study the coordination activities in LC. Specifically, we will study “Notification,” “Coordination/Scheduling,” and “Internal Phasing” emails to understand the way LC members carried out notification, moniroting, and sequencing. We should note that one specific type of coordination emails might be used for several coordination functions. For example, “Coordination/Scheduling” emails could be used

for “monitoring,” “tracking and control,” as well as “sequencing and synchronization” functions, while “Notification” emails were mainly for notification purpose, and “Internal Phasing” emails were mainly for “sequencing and synchronization” purposes.

To better understand the driving force behind the work patterns observed from the CVS log, we also need to know the nature of work LC members were doing during the time period. However, such information could not readily be obtained directly from the CVS log alone. By closely reading and comparing the emails, conference call minutes, and the “comment” information in the CVS log, plus the information provided by LC members in the interviews, I have constituted a rough timeline for LC members’ activities during the period of 1998-1999:

February-March, 1998	Benchmarking and performance improvements Alpha release
March 1998	Several rounds of bug-fixing that required full system rebuild
May 1998	Conversion to a new technical standard
June 1998	Began fill-out of LC version of library.
July 1998	Exception handling and library
September 1998	Targeted date for Beta on server
October 1998	Beta on Server
January 1999	First public release
March 1999	First round of major changes in the “middle end”
April 1999	Graphical demos run. LC system was available for downloading. Press release announced availability of LC system.
July 1999	Release 1.0a, Bug fixing (serialization, missing classes/methods)
August 1999	Release 1.0a1, 1.0a2
September 1999	Code clean-up, added copyright/legal notice

Note: Months with high volume of files submitted are highlighted in bold.

Table 3.4. Timeline of technical activities in LC (1998-1999)

(Revised from a timeline originally prepared by Xu, 2004)

In the following sections, I will try to discover the dynamics in some of the coordination-related emails in LC.

3.3.1 Dynamics in “Notification” emails

In our coding schema, “Notification” emails were defined as emails “notifying [about] a change [that] already happened or tasks completed or bugs found, e.g. update notification, bug/error notification.” LC members normally sent out an email to notify the team whenever there was any new change on the CVS server. “Notification” emails were also sent when bugs or other problems were discovered.

I compared the number of “Notification” emails and the number of files checked into the CVS server across time. The result is plotted in the following diagram:

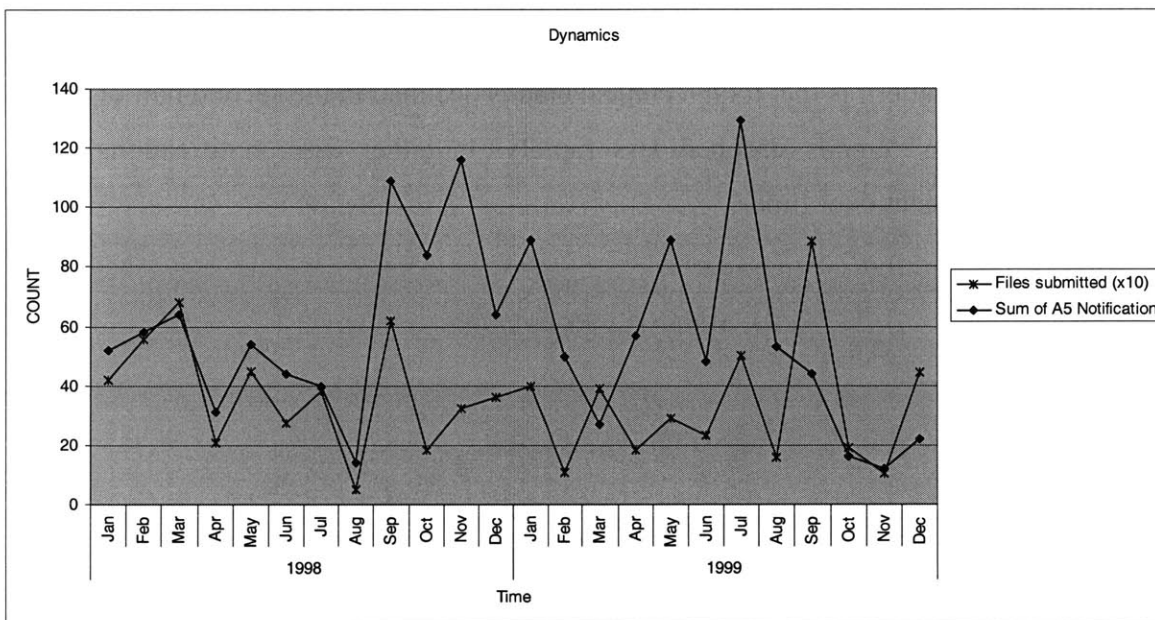


Figure 3.2. Comparison of files checked in to the CVS system and “Notification” emails (1998-1999)

The above diagram shows that there is a strong relationship between the patterns of file submits and those of “Notification” emails: they often reached peaks and valleys at the same time (e.g., March 1998, September 1998, and July 1999). This phenomenon is understandable given the way LC members used “Notification” emails in their practice.

There were also some exceptions to this relationship. For example, in March 1999, there was a peak of submitted files, while the “Notification” emails reached a low point at the same time. One reason, supported by CVS log entries, was that one CVS update usually involved multiple files, and the average number per update varied across time. So when large numbers of files were submitted in one single CVS update, we would observe a small number of CVS updates (and hence fewer “Notification” emails).

I then studied the changes in “Notification” emails during the period of study, observing that the percentage of “Notification” emails in the total technical emails was increasing in the time period. One reason, supported by a close reading of the messages, was that in later stages of the software development process, as the system design and implementation stabilized, the focus of LC members’ work shifted to bug-fixing which involved a lot of frequent fixes submitted to the server, resulting in more of both bug/error notifications and “New on server” notifications. Another reason that might also partly explain the pattern is that total technical emails declined in the second half of 1999, while “Notification” emails dropped less rapidly. In either case, notifications were increasingly prominent over time.

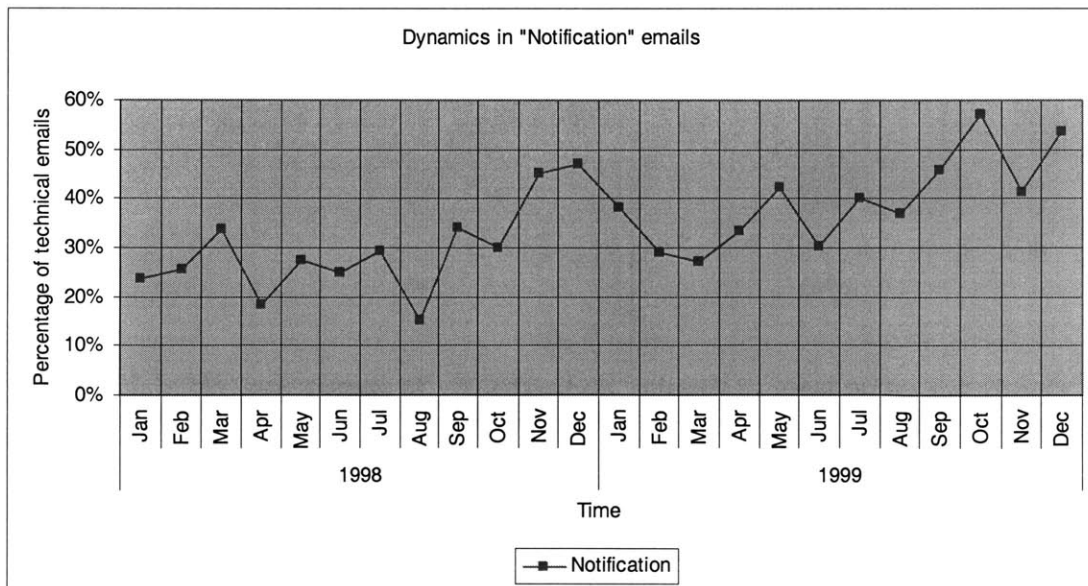


Figure 3.3. The percentage of “Notification” emails in total technical emails (1998-1999)

3.3.2 Dynamics in Coordination/Scheduling

In our coding schema, “Coordination/Scheduling” emails were defined as “a message to coordinate work (tasks) among members or to schedule meetings or events. (Status Report belongs to this category.)”

A comparison of the number of files checked into the CVS server and that of “Coordination/Scheduling” emails shows interesting patterns (see Figure 3.4.). There seemed to have been three types of relations between the two numbers: from January 1998 to August 1998, they seemed to be negatively correlated; from August 1998 to January 1999, they are positively correlated; while after January 1998, the number of “Coordination/Scheduling” emails basically kept declining towards a low level.

The technical timeline of LC (see Table 3.4.) suggests one possible explanation for this phenomenon. From January to August 1998, LC members were working more on their individual parts aiming to put the Beta version on the server in August, so there were less “Coordination/Scheduling” emails sent out; between the Beta version on the server (August 98) and the first public release (January 99), however, LC members were doing integration testing to make sure all parts worked together, necessitating more coordination activities. Two years into the LC system development, the number of coordination/scheduling emails began to decline in 1999 because LC members had developed a shared understanding of their roles and responsibilities, and therefore didn’t need to communicate as often. Another reason was they learned to coordinate more implicitly by exchanging information in technical discussion emails with fewer explicit coordination emails. Ghosh et al. (2004) observed that during this time the counts of dyadic phone calls between LC members increased, suggesting that LC members might have shifted some of the coordination functions to direct conversation over the telephone.

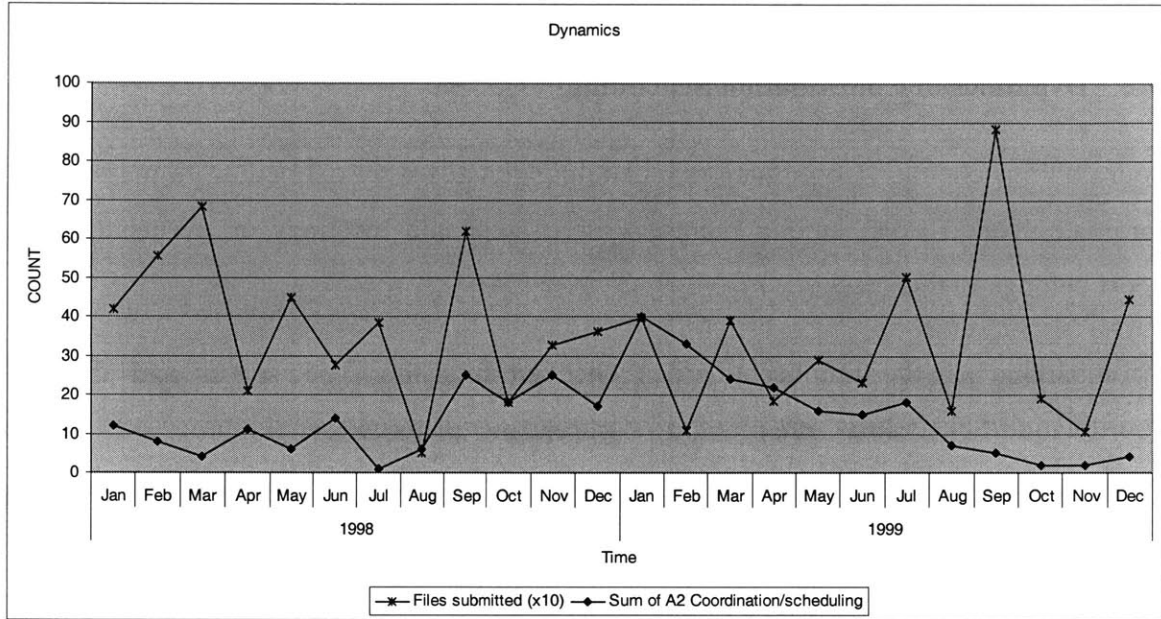


Figure 3.4. Comparison of files checked in to the CVS system and “Coordination/Scheduling” emails (1998-1999)

Figure 3.5 shows that for ten months of 1998, the percentage of “Coordination/Scheduling” messages was relatively constant. The number started to increase in November, continuing to rise until it reached a peak in March 1999. After that it decreased again to a lower level. From reading the emails and the timeline we know that around March 1999, LC members were working hard for the first public release of the LC system. The public release marked a major milestone in the LC system’s development. And it seems that LC members were doing a lot of coordination and scheduling during the period in order to complete the major release.

Another factor that might have affected this pattern was that the first round of major changes in the “Middle End” was submitted onto the CVS server at around the same time. Since the “Middle End” (done by Keith) served as the link between the “Front End” (done by Robert) and “Back End” (done by Dan), it is quite possible that the changes in the “Middle End” required a lot of coordination among all three full-time members, contributing to the March 1999 peak.

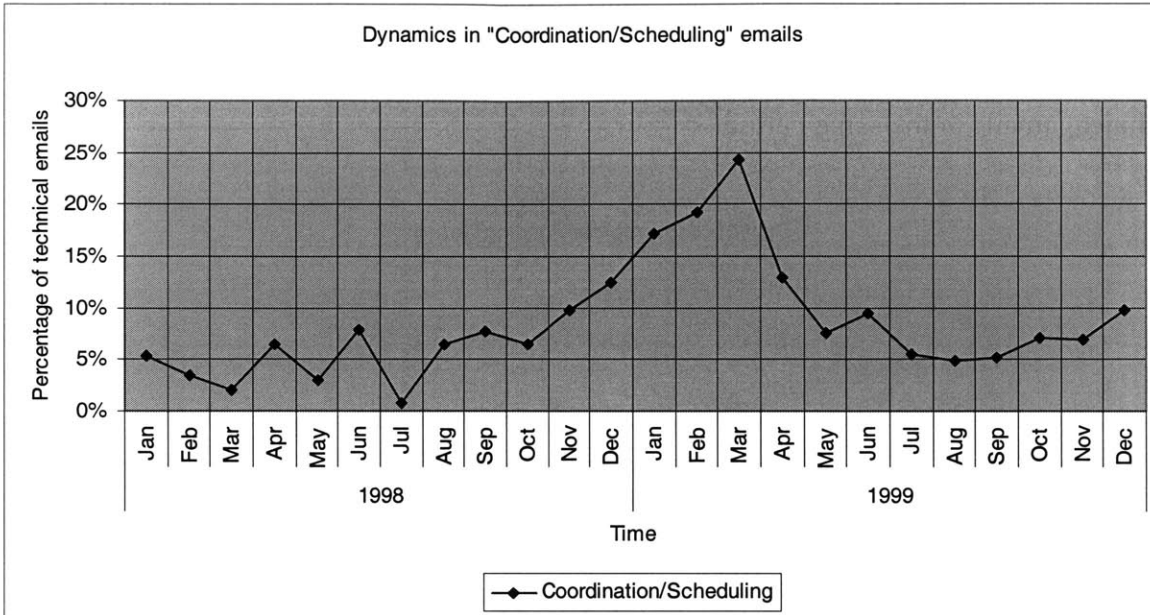


Figure 3.5. The percentage of “Coordination/Scheduling” emails in total technical emails (1998-1999)

The way the number of “Coordination/Scheduling” emails changed during the period seems to suggest that the public releases of software products are an important driving force in the software development process. And software developers need to coordinate their work more closely and schedule their work plan more frequently in order to avoid delaying the public release.

3.3.3 Dynamics in internal phasing

In our coding scheme, “Internal Phasing” emails were defined as “Reference to internal phasing for tasks. For example, deadlines and milestones set by members.” We can use “Internal Phasing” emails to study the sequencing and synchronizing of LC’s development activities.²²

Figure 3.6. shows that the number of “Internal Phasing” emails declined from January to August 1998, then jumped in September 1998, when the number reached it highest point

²² There is also an “External Phasing” category in our coding scheme. “External Phasing” emails concerns synchronizing for an external event such as a client meeting. Since this research is focused on the technical development within LC, I do not discuss “External Phasing” emails.

in the two-year period. After that, the number declined again until the end of 1999. Figure 3.7 reveals a similar pattern in the percentage of “Internal Phasing” emails in total technical emails in the same period.

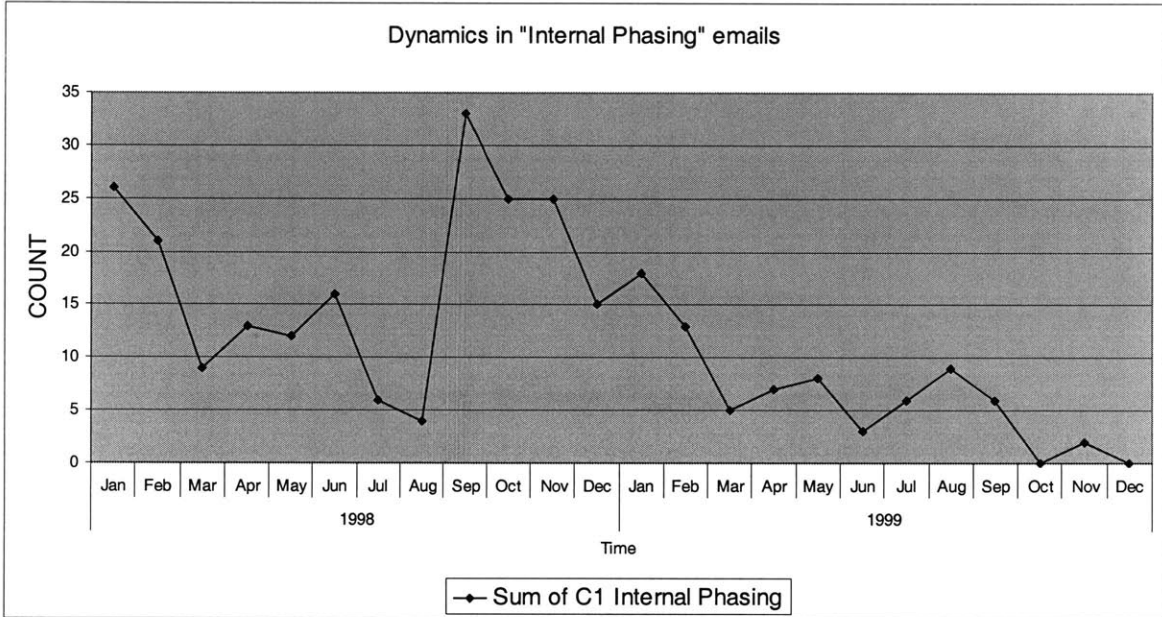


Figure 3.6. Number of “Internal Phasing” emails (1998-1999)

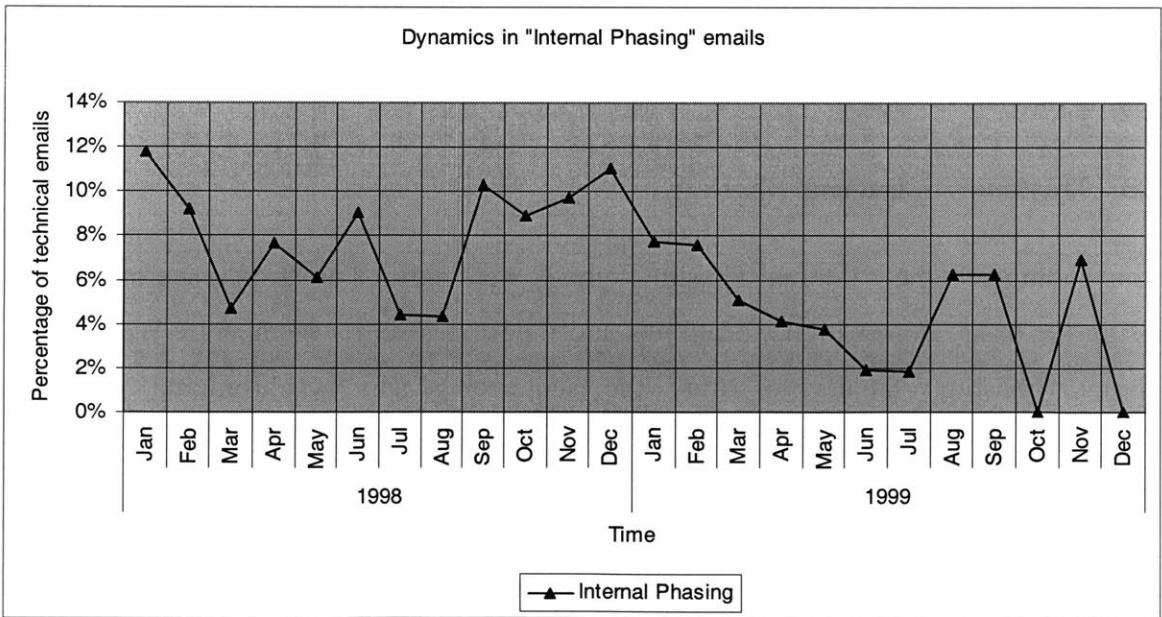


Figure 3.7. The percentage of “Internal Phasing” emails in total technical emails (1998-1999)

The number of “Internal Phasing” emails and the percentage of them in total technical emails were both evidently decreasing in the long run. There were also some short-term increases when LC members put major changes onto the CVS server: for example, there was a peak in September 1998, which was the target release date of Beta version of the LC system, and another peak in December 1998, right before the first public release of the LC system. Similar peaks could be observed at other times, such as in June 1998 when LC started to develop its own libraries for the LC system, and in August-September 1999 when they started to clean up the code and add legal notices to the files.

The fact that “Internal Phasing” emails decreased seems to suggest that LC members needed less sequencing as time went by. A possible reason is that after the focus of software development activities shifted to testing and bug-fixing. A close reading of emails shows that LC members were working in a more flexible way (what we can call the “find-a-bug-then-fix-it” pattern) instead of always working closely together to meet technical milestones (which we can call the “finish-this-part-before-this-time” pattern).

Figure 3.8 shows all the three types of coordination email together over time. It shows that notification emails came to dominate over other types of coordination emails, increasing as they fell off beginning in March 1999, after the first round of major changes in the “middle end.” Also, as we can see, the coordination emails reached a peak around January 1999, corresponding well to the first public release of the LC system.

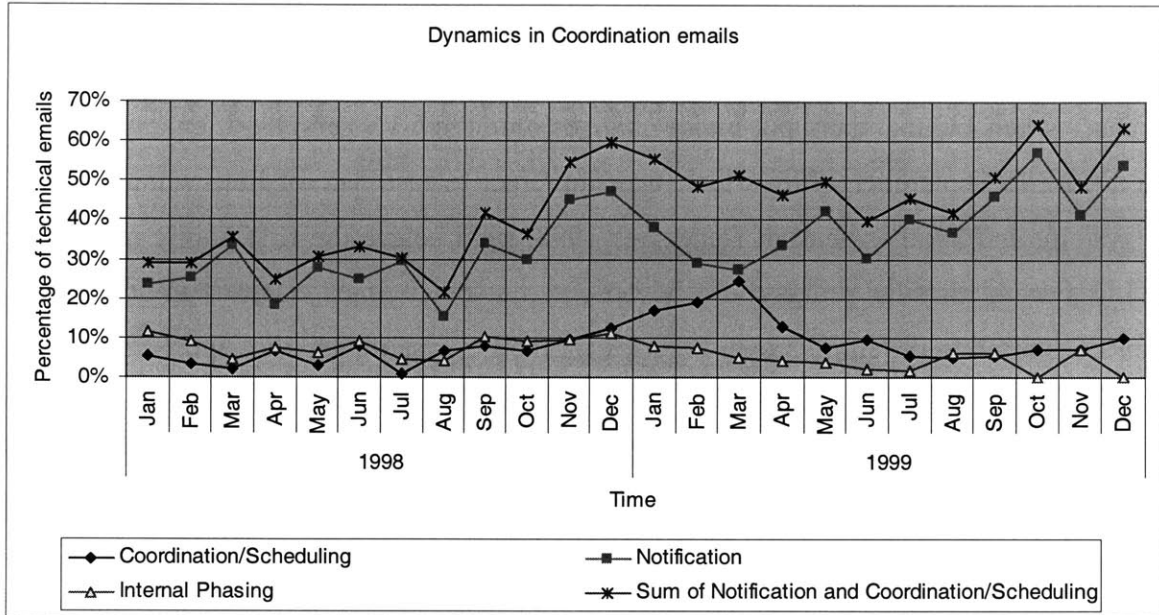


Figure 3.8 Changes of coordination emails (1998-1999)

The follow comments from Dan might partly explain it:

“...in the beginning we were doing much more design work where we had to hash things out. In the later half, all the foundations were in place, all the architecture was figured out and it was more a matter of: “Oops got a bug. Is this your bug? Talk about the bug, and go fix it.””

As Dan suggested, in early stages of software development (such as requirements analysis and system design), the majority of the communication among software developers constituted mainly of technical discussions about figuring out how to develop the product; after the software developers actually started to implement the product and do tests, the focus of their work shifted to realizing the system design and carrying out the development plan, which involved coordination through notifications of bugs found, tasks completed, and changes to the code entered through the CVS system.

Chapter 4

CONCLUSION

In this thesis, I have discussed the group characteristics, situation, task, potential conflicts, and organizational issues in a distributed software development team, and how these aspects of context together affected the coordination in the team. By focusing on the distinctiveness of the project and the team, I got a better understanding of how the team members had worked well together despite their geographical separation, time zone difference, changing market competition, and limited resources.

I also examined the team's practices and studied how their coordination was accomplished in a distributed environment. Specifically, I studied how LC members carried out the coordination functions of monitoring, tracking and control, notification, task allocation, and sequencing and synchronization in a software engineering team. With limited technology support and interaction channels, LC members based their coordination activities on a combination of information and communication technology. Specifically, they relied heavily on frequent emails and phone calls to maintain informal communications, and on the team email list and conference calls for formal communications.

LC didn't employ the formal software engineering mechanisms that are common in traditional software teams. For example, LC didn't have formal bug-tracking or peer review mechanisms but relied on the use of various type of emails to fulfill the functions of notification, synchronization, and so on. The reasons behind such choices were multi-

fold, including lack of resources and an objective to reduce work overhead. The small size and the flat hierarchy of the team, as well as the established trust, all contributed to the success of the project without many traditional software engineering techniques. For the same reasons, the coordination process was simplified and coordination costs were reduced through direct person-to-person interactions.

I then studied the role of several coordination mechanisms in LC's practice. Team email list, dyadic emails, conference calls, dyadic phone calls, and the CVS system, as well as norms and documentation, each played a distinct role in meeting specific coordination needs, and together they successfully coordinated LC members' development activities throughout the software development cycle. One impressive fact was that LC members didn't rely on advanced and complicated technical tools as their coordination mechanisms, even though they had the necessary knowledge and skills to do so. On the contrary, they used simple tools (emails) and traditional communication technologies (telephone) in fulfilling most of their coordination activities. When using a relatively advanced and complicated change management system (the CVS system) in managing the product development processes, they were also cautious and selective in making use of the available features of the system. By carefully selecting and utilizing tools based on their specific needs, LC members adopted coordination mechanisms that corresponded to LC's development processes.

This thesis also studied the dynamics of coordination activities throughout the software development cycle. I believed that since coordination was about the interdependencies among the team members (and their work), the specific coordination requirements would change at different times during the software development cycle. Understanding these changes should help to better coordinate such work by providing a combination of coordination mechanisms that serve the needs better. I observed several such patterns that suggest several patterns in the coordination of software engineering. First, coordination activities account for a large part of the communications among distributed software developers, and these activities increased during the course of development cycle; second, notifications are very important in coordinating distributed software development, especially towards the end of development process; finally, public releases of a software

product require a lot of coordination and scheduling among the distributed software developers.

By comparing the patterns of coordination emails and those of LC members' development activities on the CVS server, I also observed some relations between the coordination emails and development activities. The observations confirmed my assumption that coordination activities change dynamically throughout the software development cycle, and that carefully adapting the coordination mechanisms according to the changing coordination needs should better coordinate the development activities in both distributed and traditional co-located software engineering practices.

It is important to note, however, that this research was not comprehensive, and it is possible that other unidentified factors also contributed to the patterns. Moreover, the research findings and their implications are limited by factors such as existing relationships among the team members, the highly-similar backgrounds and expertise of the team members, and the highly informal structure in LC, which distinguished it from many other larger or more formal companies. We should thus be cautious when applying these implications to other situations.

We should also be aware that some important data (most importantly the content of dyadic phone calls, which were not recorded or transcribed) were missing from our research. The interviews make clear that the dyadic phone calls (together with emails and conference calls) played a central role in LC's operations. Though the available data were enough to study the general coordination activities within LC, it would help to understand how the dyadic phone calls complement the use of other communication and coordination tools.

Appendix A. Coding Scheme used in analyzing LC's email archive

CATEGORIES	DESCRIPTION	Percentage of total emails ²³ (1997-1999)
0. Identification	Message ID of a message	(N/A)
0.1. Already coded	A duplicate or a copy of an original message	(N/A)
A. Why (Purpose)		
A1. Initiating discussion	A message that initiates a discussion (work related) among member, for example, by posing a question, or announcing a fact or change, or providing arguments, or proposing a plan or agenda, or requesting something.	30.25%
A2. Coordinating/scheduling	A message to coordinate work (tasks) among members or to schedule meetings or events. NOTE: Status Report belongs to this category.	11.24%
A3. Discussion	A follow-up message or response to a message of category A1. But the discussion should be relevant to the work.	39.21%
A4. Report	A message to report a meeting, an event, or a fact. (e.g., meeting report (internal and external), expense report.)	0.40%
A5. Notification	Notifying a change already happened or tasks completed or bugs found. Ex. Update notification, bug/error notification.	18.03%
A6. Other	A message that does not fit into the current categories. Also make the categories as exclusive as possible, and try to code a message under one purpose category.	0.00%
B. What		
B1. Technical	Content related to the technical aspects of product or technology	64.56%
B2. Administrative	Content related to the running of the company	38.42%
B2.1. Administrative What	Description of the administrative issue	(N/A)
B3. Personal	Personal content	5.24%
B4. Other	A message that does not fit into the current categories.	0.00%
C. When		
C1. Internal phasing	Reference to internal phasing for tasks. Ex. Deadlines and milestones set by members.	9.57%
C1.1. Clock-based	Internal task-phasing based on specific times.	7.79%
C1.2. Event-based	Internal task-phasing based on specific events.	1.76%

²³ Most categories are not mutually exclusive, so percentages do not add to 100%

C2.External deadlines	Reference to external phasing for tasks, Ex. Upcoming conference, or marketing visit, or market condition.	1.40%
C2.1. Clock-based	External task-phasing based on specific times.	1.27%
C2.2. Event-based	External task-phasing based on specific events.	0.13%
C3. Other	A message that does not fit into the current categories.	0.00%
D. Work Practice		
D1.Reference to media switches	Incidents of media switches between various communication media e.g. as suggesting phone calls, reference to face-to-face or phone conversations.	9.13%
D2. Instant messaging	A message that has only the subject line (or a signature in the body content).	1.26%
D3.Reference to problems	Reference to problems (ex. Technical, procedural, communicational...)	26.21%
D4.Requesting help	A message explicitly asking help from other members to solve one's problem encountered doing one's work	3.78%
D5.Offering help	A message trying to help out other members' work (or problems)	3.25%
D6.Disagreement	Incidents of any kind of disagreement among members. (e.g., over technical decisions, work procedures, personnel hire.)	2.79%
D7.Reference to external expertise/network	Reference to external expert network for collaboration or outsourcing, e.g. soliciting or working with external experts on various tasks of product development (technical, marketing, licensing, financing...)	6.88%
D8.Negotiation (internal)	Reference to any incidents of internal negotiations (e.g. on scheduling, deadlines, technical decisions, task assignments)	5.65%
D8.1.Negotiating expertise	Negotiating each member's expertise to distribute tasks	1.10%
D9. Identity	Reference to any personal or organizational identities.	1.53%
D9. What	Description of the identity (s)	(N/A)
D10. Other	A message that does not fit into the current categories.	0.00%

Appendix B. A Day in Dan's life (Source: Ghosh et al., 2004)

A Day in Dan's Life			
Time period	Action	To/From	Conversation length (minutes)
10:18-10:23 AM	Phone	from Keith	5
10:33-10:45 AM	Phone	to Keith	12
11:17-11:20 AM	Phone	to Keith	3
11:47-11:53 AM	Phone	from Keith	6
12:08-12:12 PM	Phone	to Keith	4
12:35-12:38 PM	Phone	to Keith	3
12:42-12:44 PM	Phone	to Keith	2
1:06-1:09 PM	Phone	from Keith	3
1:17-1:20 PM	Phone	from Keith	3
2:05 PM	Code commit		
2:06 PM	Email		
2:37-3:32 PM	Phone meeting with all LC members		
3:34 -3:37 PM	Phone	from Keith	3
3:39 PM	Code commit		
3:40 PM	Email		
6:27-6:30 PM	Phone	from Keith	3
6:47-6:50 PM	Phone	from Keith	3
6:56 PM	Code commit		
6:59-7:00 PM	Phone	to Keith	1
07:00 PM	Email		
07:10-07:13 PM	Phone	to Robert	3
8:47-9:00 PM	Phone	from Keith	13
9:10 PM	Code commit		
09:11 PM	Email		
9:17 PM	Code commit		
09:19 PM	Email		
9:19-9:26 PM	Phone	from Keith	7
10:29-10:32 PM	Phone	to Keith	3
11:23-11:25 PM	Phone	from Keith	2
02:51 AM	Email		

REFERENCES

- Andres, H.P. & Zmud, R.W. (2001) A contingency approach to software project coordination, *Journal of Management Information Systems*, v 18, n 3, p 41-70
- Blau, P. & Scott, W.R. (1962) Formal organizations. Scott, Foresman, San Francisco
- Bradner, E., Mark, G., & Hertel, T. D. (2003) Effects of Team Size on Participation, Awareness, and Technology Choice in Geographically Distributed Teams, 36th Annual Hawaii International Conference on System Sciences (HICSS'03) - Track 8, Big Island, Hawaii
- Cadiz, J. J., Venolia, G., Jancke, G., & Gupta, A. (2002) All ways aware: Designing and deploying an information awareness interface, *Proceedings of the 2002 ACM conference on Computer supported cooperative work*
- Carmel, E. (1999) Global Software Teams: Collaboration across borders and time zones, Prentice Hall PTR
- Carroll, J. M., Neale, D. C., Isenhour, P. L., Rosson, M. B., & McCrickard, D. S. (2003) Notification and awareness: synchronizing task-oriented collaborative activity, *International Journal of Human-Computer Studies*, Volume 58 , Issue 5
- Chiang, R. I. & Mookerjee, V. S. (2002) A dynamic coordination policy for software system construction , *IEEE Transactions on Software Engineering*, v 28, n 7, p 684-694
- Conway, M.E. (1968) How Do Committees Invent? *Datamation*, 14 (4). 28--31.
- Crowston, K. (1997) A coordination theory approach to organizational process. *Organization Science*, 8, 2, 157-175
- Demarco, T. & Lister, T. (1987) Peopeware: productive projects and teams. New York: Dorset House Publishing
- Dewan, P., Mashayekhi, V., & Riedl, J. (2001) Infrastructure and Applications for Collaborative Software Engineering, *Coordination Theory and collaboration Technology (Ed. Olson, Malone, & Smith)* Lawrence Erlbaum Associates, p263-209
- Dourish, P. & Bellotti, V. (1992) Awareness and coordination in shared workspace, *Proceedings of CSCW '92 (Toronto Canada, November 1992)*, ACM Press
- Estublier, J. (2000) Software configuration management: a roadmap, *Proceedings of the conference on The future of Software engineering*

- Fichman, R.G. & Kemerer, C.F. (1993) Adoption of software engineering software innovations: The case of object orientation. *Sloan Management Review*, 34, 2 (1993), 7-22
- Fielding, R. T., Whitehead, E. J., Anderson, K. M., Bolcer, G. A., Oreizy, P., & Taylor, R. N. (1998) Web-based development of complex information products, *Communications of the ACM*, Volume 41 Issue 8
- Ghosh, T.; Yates, J.; & Orlikowski, W. (2004) Using Communication Norms for Coordination: Evidence from a Distributed Team, submitted to *ICIS 2004 Completed Research Papers Track on Information Systems Innovation, Usage & Impacts*
- Gopal, A., Mukhopadhyay, T., & Krishnan, M.S. (2002) The role of software processes and communication in offshore software development, *Communications of the ACM*, v.45 n.4
- Gutwin, C.; Roseman, M.; & Greenberg, S. (1996) A usability study of awareness widgets in a shared workspace groupware system, *Proceedings of CSCW '96 (Cambridge MA, November 1996)*, ACM Press, 258-267
- Gutwin, C. & Greenberg, S. (1999) The effects of workspace awareness support on the usability of real-time distributed groupware, *ACM Transactions on Computer-Human Interaction (TOCHI)*, Volume 6 Issue 3
- Hause, M. L.; Almstrum, V. L.; Last, M. Z.; & Woodroffe, M. R. (2001) Interaction factors in software development performance in distributed student teams in computer science, *ACM SIGCSE Bulletin , Proceedings of the 6th annual conference on Innovation and technology in computer science education*, Volume 33 Issue 3
- Henderson, J.C. & Coopridge, J.G. (1990) Dimensions of IIS planning and design aids: A functional model of CASE technology. *Information Systems Research*, 1, 3 (1990), 227-254
- Herbsleb, J. D. & Grinter, R. E. (1999) Splitting the organization and integrating the code: Conway's law revisited, *Proceedings of the 21st international conference on Software engineering*, p.85-95, May 16-22, Los Angeles, California, United States
- Herbsleb, J.D. & Mockus, A. (2003) Formulation and Preliminary Test of an Empirical Theory of Coordination in Software Engineering, ACM SIGSOFT Software Engineering Notes, *Proceedings of the 9th European software engineering conference held jointly with 10th ACM SIGSOFT international symposium on Foundations of software engineering*, Volume 28 Issue 5
- Homans, G. (1950). *The Human Group*, Harcourt, Brace and Company, New York

- Hunton, J.E. & Beeler, J.D. (1997) Effects of user participation in systems development: A longitudinal field experiment. *MIS Quarterly*, 21, 4 (1997), 359-388
- Im, H.; Yates, J; & Orlikowski, W. (2004) Temporal Coordination through Genres and Genre Systems, MIT Sloan School of Management: unpublished paper
- Jang, C. Y.; Steinfield, C.; & Pfaff, B. (2000) Supporting awareness among virtual teams in a web-based collaborative system: the teamSCOPE system, *ACM SIGGROUP Bulletin*, Volume 21 Issue 3
- Joseph, A. & Payne, M. (2003) Group dynamics and collaborative group performance, ACM SIGCSE Bulletin, *Proceedings of the 34th SIGCSE technical symposium on Computer science education*, Volume 35 Issue 1
- Jurison, J. (1999) Software project management: the manager's view, *Communications of the AIS*
- Karl, F. (2001) *Open Source Development with CVS*, Scottsdale, Arizona: Coriolis Group Books
- Kirsch, L. (1996) The management of complex tasks in organizations: Controlling the systems development process. *Organization Science*, 7, 1 (1996), 1-21
- Kraut, R. & L. Streeter (1995) Coordination in Software Development, *Communications of the ACM*, vol. 38, no. 3, pp.69-81
- Layzell, P.J.; Brereton, P.; & French, A. (2000) Supporting Collaboration in Distributed Software Engineering Teams, *APSEC2000: The Asia-Pacific Software Engineering Conference*, 5-8 December 2000, Singapore, IEEE Computer Society Press, pp.38-45
- Leonard, D. A.; Brands, P.; Edmondson, A.; & Fenwick, J. (1997) Virtual teams: using communication technology to manage geographically dispersed development groups, in *Sense and Respond: Capturing Value in the Network Era*, S. P. Bradley and r. L. Nolan, ed. Cambridge, MA: Harvard Business School Press
- Malone, T. W. & Crowston, K. (1990) What is coordination theory and how can it help design cooperative work system? *CSCW 90 Proceedings*: p357-369
- McCrickard, D. S.; Chewar, C. M.; Somervell, J. P.; & Ndiwalana, A. (2003) A model for notification systems evaluation—assessing user goals for multitasking activity, *ACM Transactions on Computer-Human Interaction (TOCHI)*, Vol. 10 Issue 4
- Mills, T.M. (1967) *The Sociology of Small Groups*. New Jersey: Prentice-Hall

- Perry, D.E.; Staudenmayer, N.A.; & Votta, L.G. (1994) People, organizations and process improvement. *IEEE Software* (July 1994) 36-45
- Rodden, T. (1996) Populating the application: a model of awareness for cooperative applications, *Proceedings of the 1996 ACM conference on Computer supported cooperative work*
- Shen, H. & Sun C. (2002) Group editing algorithms: Flexible notification for collaborative systems, *Proceedings of the 2002 ACM conference on Computer supported cooperative work*
- Simon, H. (1957) Models of Man, John Wiley & Sons, New York
- Simone, C.; Divitini, M.; & Schmidt, K. (1995) A notation for malleable and interoperable coordination mechanisms for CSCW systems, *Proceedings of conference on Organizational computing systems*
- Steinfeld, C.; Jang, C.Y.; & Pfaff, B. (1999) Supporting virtual team collaboration: the TeamSCOPE system, *Proceedings of the international ACM SIGGROUP conference on Supporting group work*
- Teasley, S.; Covi, L.; Krishnan, M. S.; & Olson, J. S. (2000) How does radical collocation help a team succeed?, *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, Philadelphia, Pennsylvania, United States
- Teasley, S.; Covi, L.; Krishnan, M. S.; & Olson, J. S. (2002) Rapid software development through team collocation, *IEEE Transactions on Software Engineering*, v.28 n.7
- Thomas, B. & Tilley, S., Workshop: Documentation for software engineers: what is needed to aid system understanding?, *Proceedings of the 19th annual international conference on Computer documentation*, October 2001
- Tubbs, S. (2001) A Systems Approach to Small Group Interactions, New York: McGraw-Hill
- Unger, B. W. & Walker, S. (1997) Improving team productivity in system software development, *Proceedings of the fifteenth annual SIGCPR conference*
- Van de Ven A.H.; Delbecq, A.L.; & Koenig, R. Jr. (1976) Determinants of coordination modes within organizations, *American Sociological Review* 41 (1976), 322-338
- Waterson, P.E.; Clegg, C.W.; & Axtell, C.M. (1997) The dynamics of work organization, knowledge and technology during software development, *Journal of Human-Computer Studies*, 46, 1 (1997), 79-101
- Xu, H. (2004) Managing work distribution in a software development team, MIT Sloan School of Management: unpublished paper