Coordination in Large-Scale Software Development: Helpful and Unhelpful Behaviors

Ch. Srikanth, N. Sambasiva Rao, T. Vishnu Vardhan Reddy

1 Vardhaman College of Engineering, Hyderabad
M.Tech Computer Science and Engineering,
srikanthch.tillu43@gmail.com

2 Vardhaman College of Engineering, Hyderabad,
Principal of Vardhaman College of engineering,
snandam@gmail.com

3 Vardhaman College of Engineering, Hyderabad,
M.Tech Computer Science and Engineering,,
vishnu.talla34@gmail.com

Abstract:
Software development is rarely an individual effort and generally involves teams of developers collaborating to generate good reliable code. Among the software code there exist technical dependencies that arise from software components using services from other components. The different ways of assigning the design, development, and testing of these software modules to people can cause various coordination problems among them. We claim that the collaboration of the developers, designers and testers must be related to and governed by the technical task structure. We propose a Collaborative Work allocation Patterns to handle these collaboration practices. We propose a methodology to define collaborative work allocation patterns that can be used in order to handle the collaborative software development coordination issues.

Keywords: Software development, Collaboration, Software code, Patterns.

1. Introduction:
Coordination between software development teams is one of the most difficult-to-improve aspects of software engineering. Kraut and Streeter argue that the software industry has been in crisis mode for its entire existence, and a root cause is the difficulty in coordinating work between teams of developers. Researchers have studied professional software development teams empirically to gain greater understanding of how software development processes, tools, and people impact coordination. The importance of intra and inter team coordination is a foremost concern as software development increasingly becomes globally distributed, and remains a persistent challenge in other disciplines as well. We view coordination as decision-making and action requiring communication, capacity and cooperation.

These three components of coordination are necessary, but by themselves insufficient, for coordination to take place. Communication is necessary because person A needs to communicate to person B, in some form, what needs to be done, and B needs to understand the communication. Capacity is necessary because B needs to be able to do what is
required of them. Cooperation is necessary because B needs to be willing to do what is required of them. If any of the three necessary components are lacking, the outcome will be less than ideal. Viewing coordination through this framework leads us to ask several research questions:

1. What kinds of behaviors are associated with being helpful or unhelpful to others?
2. How do individuals on a software team communicate to get work done?
3. How do software teams manage dependencies on a personal level?

To understand inter- and intra-team dependencies in large scale software development, we conducted a large interview-based study of a 300-person Microsoft software development group, of which two of its teams are distributed globally. We collected 31 hours of interviews from 26 engineers to learn with whom these individuals collaborated and which actions the individuals considered helpful or unhelpful for coordination. The interviews were audio recorded, transcribed, and coded. We corroborated our interview data with a follow-up survey of 775 Microsoft engineers. The interviews revealed that the most helpful behaviors between all of the teams are related to capacity, cooperation and availability. The unhelpful behaviors are related to location (primarily felt by the two distributed teams), ownership and awareness, capacity and availability.

We find that problems of coordination in the distributed teams can be considered a superset of the problems of collocated teams; distributed development adds additional difficulties caused by location (i.e. time zone), email, culture, and meetings. The survey data supports the results of the interview survey; many of the problems perceived by the interviewed team members are also seen in other groups at Microsoft. We use our findings to provide a system-level perspective of the team's network of task-based interactions, organized by role and location, which highlights the effectiveness of the team's relationships. Based on our findings, we suggest concrete actions that can be taken to improve coordination between software teams.

Our view of coordination helps reveal how communication, capacity, and cooperation interplay to affect successful software development projects. Our study finds that many of the behaviors that impact engineers most are not directly technical issues, such as code and APIs, but rather coordination issues. These are commonly referred to as "human aspects" of software development, and they appear, from the data gathered in this study, to be highly significant to the software engineer's tasks. We show that taking a systems view of the situation reveals unanticipated asymmetries in the relationships between roles, and highlights interactions that may benefit most from improvement. We propose several process and tool changes to address each aspect of our coordination model, that, when enacted, are predicted to improve interaction effectiveness. A new study lies ahead to measure the effects of our changes.

2. Supporting Articulation Work Using Software Configuration Management Systems:
Empirical studies of groupware technologies hold the promise of helping us to understand the reasons why systems fail in practice and some of the ways that we can make them work (Grudin, 1989; Okamura et al., 1994; Grudin and Palen, 1995). One challenge that researchers interested in empirical studies face is finding organizations using groupware technologies. As more groupware technologies become commercially available
a number of studies have begun to examine the how groupware works in practice (Orlikowski, 1992; Bowers, 1994). Many of the empirical studies to date have focused on the initial adoption and use of these groupware technologies among small groups of users. It is harder to find groups of users who have been working with groupware for such a long time that the systems have become commonplace in their work.

Software product development provides a promising domain to find groupware like systems that have been in use for some time for several reasons. First, most currently available commercial software packages were built collaborative by teams of developers. Second, development organizations already have the necessary technical infrastructure to support groupware systems. Third, developers have a good base of technical skills and an inclination towards learning and adopting computer software. Fourth, academic researchers and commercial organizations have tried to support the development process by building tools to support the work since the mid-1980s.

This paper describes an empirical analysis of one technology that attempts to support collaborative software development work: configuration management (CM) systems. It examines the coordination mechanisms that the tool provides and how they interact with the work of software development. The paper begins with an overview of CM systems, their groupware-like properties, the organizations studied and the methods used to gather and analyze the data collected. Then I introduce the three aspects of the support that the tool provides: the challenges of representing the work, the need to support both individuals and groups working together, and how the assumptions about software development built into the tool interact with others in the organization. The following section discusses the implications that these observations have for CSCW research.

In this paper I examined the role of CM systems in coordinating software development work. Instead of providing enhanced or increased communicative capabilities the tool supplies coordination mechanisms. In this study I have explored three aspects of those coordination mechanisms: representation issues, the need for different levels of mechanisms, and how those mechanisms support a model of software development.

This study offers two sets of conclusions, for the designers of CM systems, and for CSCW researchers. The design and development of most CM systems available today has been influenced by the difficulties of controlling the evolution of software. CM specialists have only recently realized that their systems have groupware-like features (Nix, 1994). The developers of future systems could learn from the existing empirical studies of all groupware technologies about the challenges of designing usable and useful systems.

However, this study offers a more substantive conclusion applicable to CM specialists and CSCW researchers. All three of the aspects highlighted the interplay between the organization and the tool that goes on continuously during the use of the tool. The challenge of representing complex merges is that of including organizational context inside the tool by making more information about the development process available to developers. The organization also coordinates software development efforts when the tool provides no support. Finally, the organization where the tool is deployed, and outside professional institutions, provide models about software development that developers must reconcile in their daily interactions with the tool itself. This study suggests that organizational analysis (see Morgan, 1986) can be usefully
applied to empirical studies of groupware technologies.

Understanding the interaction between the organization and the tool is important for designers building CM tools and other groupware systems, and CSCW researchers interesting in understanding the long-term use of collaborative technologies. To date most of the empirical studies have focused attention on interactions among the individuals using the tool in their work. Another stream of research that holds promise for understanding how groupware technologies support the work of organizations consists of understanding the organizations themselves and how they interact with groupware technologies.

3. An Empirical Study of Global Software Development: Distance and Speed:
Communication and coordination issues in large software engineering projects have always been formidable. Increasingly, engineers and managers must add the challenges of coordinating work across sites, spanning national, language, and cultural barriers. Driven by market and resource requirements, the push toward globalization has generated a wide variety of problems for software developers.

Previous research, suggests that cross-site communication and coordination issues cause a substantial loss of development speed. In this paper, we investigate relationships among delay, communication, coordination, and geographic distribution of work, in order to shed light on the possible mechanisms responsible for introducing delay.

A major software development effort at Lucent Technologies has been distributed for several years. It began with two sites in two countries in one continent. It has now grown to six primary sites located in four countries, in two continents, with an additional seventh supporting site in a third continent. This paper reports how problems with delays impacted this organization, and how the understanding achieved from this empirical research informed the development of tools to overcome these problems.

In the remainder of this introduction, we review literature on distributed work and how it differs from collocated work. In the following section we briefly describe our empirical methods. Then, we present new results on communication patterns across and within sites, and results showing the relationship of cross-site work, delay, and other important variables. Finally, we draw out the implications of these observations for achieving success in cross-site work, and conclude the paper.

4. Communication and distance:
In sharp contrast to the popular image of software developers as relatively introverted and secluded, they in fact spend a large proportion of their time communicating. For example, in an empirical study of time use by developers in a large software engineering organization, Perry, Staudenmayer, and Votta reported that “one of the most salient impressions conveyed by observation was the sheer amount of time each developer spent in informal communication” (p. 41). The developers in their study spent an average of 75 minutes each day in “unplanned interpersonal interaction.”

In an 8-month study of a medium-sized telecommunications software project [8], an analysis of time sheets indicated that about 50% of time was spent in “group work” (meetings and unplanned work-related discussions) during the first month, and this level dropped fairly steadily until only about 10% of time was spent in group work during the last month. Design activities, in particular, seemed to require a very large proportion of collaborative work (over 50% in all but one 4-week period), in
contrast to the relatively solitary activities of coding and testing.

In contrast to the frequent interaction of co-located work, there is very convincing evidence that the frequency of communication generally drops off sharply with physical separation among co-workers’ offices, and that the sphere of frequent communication is surprisingly small. Tom Allen, in a study of engineering organizations, reported that the frequency of communication among engineers decreased with distance. Further, he noted that when engineers’ offices were about 30 meters or more apart, the frequency of communication dropped to nearly the same low level as people with offices separated by many miles.

Kraut et al [5] found similar results for scientists. Further, they found that the rate at which scientists collaborated spontaneously with one another was also a function of distance between offices, and that this effect was more powerful than the effect of same-discipline scientists tending to collaborate more frequently with one another. Presumably, the more frequent communications led to conversations in which common interests were discovered and acted upon.

These findings are particularly troubling in rapidly evolving, high technology environments, where the competitors, products, standards, and customers routinely create a demand for significant, unforeseen changes in requirements throughout the development cycle. In organizations with rapidly changing environments and “unstable” projects, informal communication is particularly important. For example, as requirements change, it is hard for the formal mechanisms of communication, such as specification documents to react quickly enough. Often news of change, its significance, and its potential impact, is propagated informally and very quickly among the development staff. Under such conditions, the pattern of lateral communication across sites should be particularly important in the development organization under study.

Research showing the importance of informal communication has lead to a variety of technologies designed to stimulate casual conversation among workers at different sites. These technologies have included video, audio, and text. To this list we must now also add instant messaging, a technology that has spread very rapidly, and is beginning to infiltrate the workplace. We have seen little indication, however, that these technologies have been widely deployed in software engineering organizations.

These observations about communication and distance also highlight the importance of understanding the dependencies among the various kinds of work involved in software development. In a study of six software engineering organizations, Grinter, Herbsleb, and Perry observed four different ways of organizing work across sites that evolved within a single global corporation. Each represented an attempt to minimize requirements for cross-site communication in the context of particular types of product architectures and mechanisms for coordinating work. There are also indications, from a study of an automotive engineering group, that, where possible, engineers will try to reduce the coupling of cross-site work.

In a case study of a software engineering organization spread across several sites, Herbsleb and Grinter investigated how the organization used a number of mechanisms, including plans, processes, and interface specifications, to coordinate the cross-site work. Each mechanism, however, was vulnerable to imperfect foresight and unexpected events,
which required substantial communication to coordinate activities and renegotiate commitments. Despite the need for communication, there was a nearly total absence of informal, unplanned communication across sites.

The difficulties of knowing who to contact about what, of initiating contact, and of communicating effectively across sites, led to a number of serious coordination problems. Among these problems were unrecognized conflicts among the assumptions made at different sites and incorrect interpretation of communications. The most frequent consequence of cross-site problems was delay in the resolution of work issues. By delay, we mean the additional time it takes to resolve an issue when more than one site is involved. So, for example, if a part of the design or code needs to be changed, or if someone needs a better understanding of how some part of the product works, people at more than one site may need to be involved in information exchange, negotiation, and so on, in order to find a solution. Such issues arise very frequently in software development.

Delays in resolving work issues can slow development considerably. Issues that would typically be resolved in hours or minutes often stretched out to days or weeks in the effort to find, establish contact, and have the necessary collaborative sessions with the right people to achieve resolution.

Qualitative studies, have shown how individuals are disrupted by cross-site coordination challenges. But questions remain about the cumulative effects, for example, how distance affects the speed with which software engineering tasks are accomplished, and how distance is related to other important variables that influence speed, such as the size of a task, or the number of people involved. In addition to being important research questions, these are critical pragmatic issues as businesses become more globally distributed. Speed to market has become the most critical factor for succeeding with new products.

In this paper, we use two independent sources of data to examine the effect of distributed work on speed, and then examine a number of properties of cross-site versus same-site communication that may account for these differences. Finally, we discuss the implications of these findings for tools to address these communication issues.

5. Research questions:
This paper reports a study of one geographically distributed organization, with particular attention to the effects of geographic distribution on delay in the development life cycle. We also examine the patterns and quality of communication in order to shed light on possible causes of delay.

Does cross-site work introduce delay, as compared to same-site work? Previous research suggests that working across sites introduces substantial delay because of reduced communication, difficulty in finding the right person and establishing contact, as well as having an effective collaborative session. We examine quantitative data comparing the time required for similar same-site and cross-site work.

What factors influence the time interval required to make a software change? What role, if any, does spreading work across multiple sites play in lengthening this interval? Assuming that there is an association between multi-site work and longer intervals, there are many distinct ways in which working across multiple sites might introduce this delay. By modeling the time interval required to make a software change, we extract evidence helpful in determining the nature of the relationship, and what causal mechanisms are plausible.

What differences are there between
same site and cross-site social “networks” and their effectiveness? Are they stable over time? One of the possible causes of multisite delay is the difficulty of communication and coordination inherent in cross-site work. In order to begin to understand this issue, we address several basic questions about communication within a site and how it differs from communication across sites. For example, what is the relative size of local and cross-site social networks? How much instability is there in these social networks once they are established? Is there a perception of greater misunderstanding of tasks, priorities, plans, and changes across sites?

Speed is perhaps the single most important success factor in modern high technology businesses. In the face of the growing globalization of all facets of work, from virtual teams to virtual corporations, the demand for speed must be accomplished, if it is to be accomplished at all, in a geographically distributed environment.

It is important to keep in mind that these results might differ for organizations that distribute work across sites indifferent ways, or that use other business arrangements, such as outsourcing. The possible effects of such differences are open questions for future research.

Does cross-site work introduce delay, as compared to same-site work? The answer would appear to be “yes, but indirectly.” Both the survey data and the MR data indicate that work that spans sites takes longer than work that does not cross sites. The fact that both sources of data indicate substantial cross-site delays increases our confidence in this result. But the nature of this relationship is not necessarily simple.

What factors influence the time interval required to make a software change? What role, if any, does spreading work across multiple sites play in lengthening interval? A graphical model showed that size, diffusion, and number of people were all directly related to MR interval. It does not appear to be the case that cross-site changes take longer simply because they are larger, or touch more modules. Rather, multi-site changes tend to involve more people changing the code than do comparable single-site changes, and number of people is in turn strongly related to length of interval. This suggests that multi-site communication, coordination, and/or social networks may differ from their single-site counterparts in a way that requires more people to participate, thereby introducing delay.

What differences are there between same-site and cross-site communication networks and their effectiveness? Are they stable over time? We identified several differences, including the size of the communication network, the difficulty of finding people, the reduced likelihood of getting timely information, the clarity of plans, and the reduced likelihood of receiving help with heavy workloads. Both remote and local social networks were extremely volatile, with about 80% change in seven months.

Interestingly, and perhaps ironically, the only factor statistically related to the length of delays (as measured by the surveys) was the reported lower levels of receiving help with especially heavy workloads. The irony is that this is one factor where we have data indicating that no one seems to perceive him/herself as part of the problem. In other words, people believe they are no less helpful to remote colleagues, but the same population of respondents reports they are less likely to receive help from remote, as compared to local, colleagues.

There are several possible explanations of this effect. One simple explanation is that local networks are larger, hence one receives more help locally. The
fact that answers to the parallel question about help rendered by the respondent are not different for local and remote colleagues casts doubt on whether their responses are influenced by network size.

Alternatively, it may be that people attempt to be equally helpful, but help offered across sites is relatively ineffective, so the perception of the recipient is that little help has been rendered. Or it may be that it is difficult to accurately convey the urgency of a situation across sites, so the potential helper believes he/she is offering an appropriate level of help, but underestimates the problem. These possibilities have somewhat different implications for collaboration tools, since the first would require tools that help with effectively carrying out the work, while the latter places more importance on communication tools that help convey more context, and perhaps more emotional content. In any case, this is a particularly urgent problem since it is directly related to delay, and the responses suggest that people in general will see no need to take any action.

6. Multiple sites, number of people, and delay:
In general, the survey results suggest that communication and coordination are substantially disrupted across sites. There are several possible ways in which this reduced capacity for effective communication could result in the need for more people to accomplish work that could be carried out by fewer co-located people: Introducing conflicts. It is much easier to design and implement software changes in such a way as not to conflict with other parts of the product when all relevant parts are local. As a result of countless informal interactions, developers are much more aware of the details of pieces of products built locally, and are much more likely to communicate with those who work on them. This greater awareness of the context in which they are working allows developers to avoid creating conflicts with existing code, or with other changes.

Informal participation in changes. It is relatively easy to consult with local developers who have responsibility and expertise for code related to the code being changed. It may be that after such informal consultation, the developers primarily responsible for the change also make the necessary changes in the related parts of the code. This informal, and presumably faster, means of implementing changes may not be possible across sites, where the necessary communication and trust are absent, and a more formal process must be followed, involving more people submitting code for the change.

Getting right people involved. It is much more difficult to know precisely who has what responsibilities and expertise across sites. It is also much more difficult to communicate precisely what the relationship is between a proposed change made to code at one site and the impact of that change on code at another site. Given these constraints, it is much less likely, when changes at one site necessitate related changes at another site, that the correct person is involved from the beginning. Compared to the single-site case, it is more likely that the first person will make some changes as requested, only later realizing that slightly different expertise is required. If it is more difficult to get the right people involved from the beginning, changes will presumably involve more people and also take longer. We leave it to future research to distinguish among these and other possibilities. Regardless of the precise mechanism, tools and practices that increase the ability to find experts, communicate clearly, and to increase informal communication, should reduce these problems.
7. References:


