Coordination Mechanisms in Geographically Distributed Software Development

J. Alberto Espinosa
E-Mail: alberto@american.edu
Tel. 202.885.1958 Fax. 202.885.1992
American University
4400 Massachusetts Avenue, N.W., Washington, D.C.

Sandra A. Slaughter*
E-Mail: sandras@andrew.cmu.edu
Tel. 412.268.2308 Fax. 412.268.7345
Tepper School of Business
Carnegie Mellon University
Tech & Frew Streets, Pittsburgh, PA 15213

James D. Herbsleb
E-Mail: jdh@cs.cmu.edu
Tel. 412.268.8933 Fax. 412.268.7287
Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213

Robert E. Kraut
E-Mail: robert.kraut@cmu.edu
Tel. 412.268.7694 Fax. 412.268.1266
Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213

December 28, 2005

*** Contact author**

Proceedings of the First International Conference on Management of Globally Distributed Work
Bangalore, India.

Coordination Mechanisms in Geographically Distributed Software Development

Abstract

Despite technological advances in software engineering and collaboration tools, coordination in large-scale, geographically distributed software development continues to be problematic. Traditional theories suggest that collaborators coordinate by programming their tasks and by communicating with each other, but recent research also suggests that they coordinate implicitly through team cognition. However, there is very little empirical evidence on how the use of these various coordination mechanisms affects coordination effectiveness in the context of distributed work. This study investigates coordination in large-scale software development in a major telecommunications firm where the developers are geographically dispersed. Our findings indicate that shared knowledge of team members and use of software configuration management tools are associated with greater coordination effectiveness, while other coordination mechanisms including shared knowledge of the task, general communication, telephone communication and other software tools are not. Further, the use of software configuration management tools has the greatest positive impact on coordination, and developers use the tools not only for task programming but also for communicating about their work. We discuss the implications of our findings for coordination in large-scale, geographically distributed software development.

Key Words: Software Development; Coordination Effectiveness; Coordination Mechanisms; Geographically Distributed Work

Coordination Mechanisms in Geographically Distributed Software Development

Introduction

Large-scale software development involves a large number of teams and individuals working in parallel on different parts (e.g., files, modules, sub-systems, etc.) of the same software product, a product that often contains more than a million lines of code. Because of the complex dependencies of large software products and processes, and the time pressures imposed by deadlines and competition, effective coordination in this context is of great importance. Most of the work that goes into developing large-scale software products is done individually and later integrated with the work of others, thus making this collaborative effort primarily an asynchronous task requiring a fair amount of coordination. In addition, because it is generally difficult to find all of the necessary human resources in one location, large-scale software development often involves people collaborating from more than one location. The ability to span wider geographical areas has many potential benefits, including proximity to clients, and access to special talent and technical resources (Carmel 1999). However, studies are finding that geographic dispersion brings increased coordination overhead and more substantial delays in software development (Herbsleb, Mockus et al. 2000). Thus, it is very important to identify which factors contribute to coordination effectiveness in this domain, and how distance affects a software team's ability to coordinate.

Traditional organization theories suggest that coordination is accomplished by organizing (i.e., programming) tasks and by communicating (March and Simon 1958; Thompson 1967; Van de Ven, Delbecq et al. 1976), but recent research suggests coordination can be facilitated using cognitive mechanisms like shared task knowledge (Cannon-Bowers, Salas et al. 1993; Klimoski and Mohammed 1994; Kraiger and Wenzel 1997) and knowledge about each other (Wegner

1995; Faraj and Sproull 2000). However, the joint use and effects of these three types of coordination mechanisms – task programming, communication and cognitive – have not been empirically investigated in geographically distributed teams. We would expect that geographically dispersed teams use different portfolios of task programming, communication and cognitive mechanisms to coordinate. Because distance separation provides fewer opportunities for interaction and acquisition of shared knowledge, it will be more difficult for distributed teams to communicate effectively. Thus, geographically dispersed teams may rely more heavily on task programming mechanisms such as software tools and configuration management systems to coordinate their work.

While the importance of coordination in software development is being addressed in the literature, to the best of our knowledge, there are no studies that have specifically investigated the joint effect of these three main types of coordination mechanisms in geographically distributed teams. More specifically, the ***research question*** of this study is:

> *How effective are task programming, communication and cognitive mechanisms in helping teams coordinate in large-scale, geographically distributed software development?*

Theoretical Foundations

Teams are important units of work in organizations (Hackman 1987; Sproull and Kiesler 1991). They bring diverse expertise, skills and resources to complex tasks that may be too large or complex for a single individual to undertake. However, when sub-tasks have substantial dependencies, larger teams also become more difficult to coordinate because there are more possible lines of communication and dependencies among members. Coordinated teams manage these dependencies effectively using a number of mechanisms and processes. Teams coordinate explicitly using task programming mechanisms (e.g., schedules, plans, procedures, etc.) or by

communicating (e.g., orally, in writing, formally, informally, interpersonally, in groups). We call these mechanisms "explicit" because team members use them purposely to coordinate.

However, teams can also coordinate "implicitly" (i.e., without consciously trying to coordinate) through team cognition mechanisms, which are based on the shared knowledge team members have about the task and about each other. This shared knowledge helps team members understand what is going on with the task, thus helping them anticipate what is going to happen next and which actions team members are likely to take. Members can use this knowledge to plan their own actions, thus enabling them to become more coordinated. Because dependencies can be managed in more than one way, teams use a mix of coordination mechanisms to manage task and member dependencies. Consequently, it is important that we understand how these coordination mechanisms complement and interact with each other.

Furthermore, when teams are geographically dispersed, distance separation provides fewer opportunities for spontaneous interaction and acquisition of shared knowledge. Team communication and information exchange in asynchronous teams are less interactive, thus taking longer for members to get an acknowledgement, obtain an answer, or make reparations in their communication. This is why most asynchronous teams periodically schedule face-to-face meetings (e.g., debriefing meetings, staff meetings). Conversely, synchronous teams (e.g., flight crews, surgical teams) can communicate interactively and exchange information more efficiently. Distance separation also makes it difficult for the team to interact and exchange task knowledge. Interaction in distributed teams is mediated by technologies that have less rich communication media than face-to-face, thus making it more difficult to share and integrate knowledge. Furthermore, distance separation makes it more difficult to get to know colleagues and figure out when they are around. Consequently, shared knowledge is more difficult to develop in asynchronous and distributed teams. At the same time, because the communication

effectiveness among members is impaired, having previously developed shared knowledge of the task and team could potentially be of great benefit. Unfortunately, there is not much empirical research that has investigated the joint effects of task programming, communication and team cognition in geographically distributed contexts, thus the importance of conducting research in this area. This research tries to fill this gap.

The theoretical foundations for this study integrate the classic organizational research literature, which focuses on explicit coordination mechanisms (i.e., task organization and team communication), with more recent research on team cognition, which focuses on implicit coordination mechanisms (i.e., shared knowledge of the task and team). We begin by defining coordination, which is the central piece of the framework. We then consider how coordination mechanisms affect coordination effectiveness in the context of geographically distributed work.

Coordination Defined: Managing Dependencies

Consistent with the research literature on coordination theory we define coordination as "the effective management of dependencies among sub-tasks, resources (e.g., equipment, tools, etc.) and people" (Malone and Crowston 1990; Malone and Crowston 1994). If things can be done independently, then there is no need to coordinate work activities. Conversely, when multiple individuals need to interact in a synchronized fashion to carry out their task activities, there are dependencies that need to be managed. Team members can perform their individual responsibilities competently and still be uncoordinated with the rest of the team if the respective dependencies among their sub-tasks are not properly managed. Coordination can be viewed as both, a process (i.e., coordinating) and an outcome (i.e., state of coordination). The process of "coordinating" can be defined as the activities carried out by team members when managing dependencies. For example, when a software team convenes a debriefing meeting so that all parties can share the status of their respective tasks, they are managing the dependencies they

face by communicating with each other about what is going on with their individual task assignments. Coordination as an outcome (i.e., state of coordination or coordination success) can be defined as the extent to which dependencies have been effectively managed. For example, a software team will be highly coordinated if all relevant software parts for a given project integrate and work well together, are delivered on schedule, and are produced according to the software process in place (i.e., technical, temporal and software process dependencies have been effectively managed).

Traditional Coordination Mechanisms: The Classic Organizational Theory View

The management of task dependencies is accomplished via coordination mechanisms. Traditional coordination mechanisms and processes have been studied in the classic organizational research literature for many years. This literature suggests that team members coordinate via task organization mechanisms or by communicating with each other (March and Simon 1958; Thompson 1967). March and Simon suggested that the coordination of repetitive and routine aspects of the task can be "programmed" using mechanisms like tools, schedules, plans, manuals and specifications. Some have used other labels for these mechanisms, such as "task programming mechanisms" (March and Simon 1958), "impersonal mechanisms" (Van de Ven, Delbecq et al. 1976) and "administrative coordination" (Faraj and Sproull 2000).

Traditional mechanisms should facilitate coordination in geographically distributed software development work. Project plans and schedules, for example, help manage temporal dependencies because developers can find out when they need to deliver their respective software parts, or when can they expect to receive finished parts from other developers, so that subsequent software activities (e.g., testing, integration) can be carried out in a timely manner. Similarly, modern software configuration management systems have features that enable developers in different physical locations to work simultaneously on different parts of the code

without interfering with each other. The configuration management system helps teams to coordinate because it facilitates update and integration of code, and protects parts of the code that are affected by a modification, so that multiple developers working on the same code do not interfere with each other. Thus, we would expect that:

H.1: *Task programming mechanisms have a positive effect on coordination effectiveness in large-scale, geographically distributed software development*

While task programming mechanisms are very effective in helping coordinate those aspects of the task that can be programmed, these mechanisms may be less effective for more uncertain aspects of the task because dependencies cannot be managed in a programmed way. March and Simon suggested that teams resort to communication (i.e., "coordination by feedback") in these cases. The literature has also used different terms for this type of mechanism like coordination by "mutual adjustment" (Thompson 1967) and "personal" and "group" mechanisms (Van de Ven, Delbecq et al. 1976). Such mechanisms should also facilitate coordination in geographically distributed software development. For example, communication can help the team coordinate if several delivery deadlines have already been missed or if there has been a major hardware failure that calls for re-scheduling. When these things happen the team needs to communicate to cope with the changing situation or to figure out which other task organization mechanisms are more suitable to the new situation. Thus, we would expect that:

H.2: *Team communication has a positive effect on coordination effectiveness in large-scale, geographically distributed software development*

Implicit Coordination Mechanisms: The Team Cognition View

Recent research in team cognition suggests that as team members interact with each other and gain expertise with the joint task, they develop knowledge that helps them coordinate implicitly (Cannon-Bowers, Salas et al. 1993; Klimoski and Mohammed 1994; Levesque, Wilson et al. 2001). Such implicit coordination has been referred to as the "synchronization of

member actions based on unspoken assumptions about what others in the group are likely to do" (Wittenbaum and Stasser 1996). For example, transactive memory is knowledge of who knows what in the team. This knowledge helps members coordinate because they know whom to contact when they need information, and also because members develop expectations about who in the team will acquire which kinds of new information when the member joins the team (Liang, Moreland et al. 1995; Wegner 1995).

The literature on team cognition argues that team members develop shared knowledge about many things (e.g., goals, strategies, processes, team interaction, etc.), but that most of the knowledge that matters to work teams relates to either task work (i.e., activities necessary to carry out the task) or team work (i.e., activities necessary to work with each other) (Klimoski and Mohammed 1994; Rentsch and Hall 1994; Cooke, Salas et al. 2000). Consequently, it helps to make a distinction between shared knowledge about the task (i.e., shared mental model of the task) and shared knowledge about team members (i.e., shared mental model of the team).

Having shared knowledge about technical concepts, products and processes can help team members develop accurate expectations about future states of the task and improve common grounding in their communication, all of which helps coordination. For example, a study of seventeen software design teams for large systems found that one of the most salient problems leading to mistakes and the need for additional effort was the thin spread of application domain knowledge within the team (Curtis, Krasner et al. 1988). The study concluded that knowledge sharing and integration in this domain was necessary to ensure positive outcomes in this domain. A later case study of a software design team concluded that knowledge acquisition, sharing and integration were critical in this domain (Walz, Elam et al. 1993).

Similarly, having shared knowledge about members of the team can also help coordination because individuals can develop accurate expectations of what other teammates

know and how they may act in particular circumstances, thus helping them plan their own actions. For example, a study with large-scale software developers found that knowing where expertise resides in a team had a positive effect on team performance (Faraj and Sproull 2000). Another study with software requirements analysis teams at two separate organizations found that team members that exhibited a "collective mind" were more coordinated because individuals came to understand how their work contributed to the work of the group. The study concluded that it was important for software team members to develop models of what others in the team did (Crowston and Kammerer 1998) which can help team performance in complex, non-routine, and high-reliability tasks (Weick and Roberts 1993). Therefore, we would expect that:

H.3: *Implicit coordination mechanisms based on team cognition – shared knowledge of the task and of team members – have a positive effect on coordination effectiveness in large-scale, geographically distributed software development*

Geographic Dispersion and the Effectiveness of Coordination Mechanisms

There is empirical evidence that geographic distance makes it more difficult to coordinate tasks in software development (Curtis, Krasner et al. 1988; Herbsleb and Grinter 1999; Herbsleb, Mockus et al. 2000) because geographically distributed collaborators have fewer opportunities to interact (Kraut and Streeter 1995) and because their communication is less effective (Curtis, Krasner et al. 1988). In addition, distributed teams are often larger because of the need for counterpart and liaison roles between sites (Herbsleb, Mockus et al. 2001). As the discussion in this section suggests, a team will employ a mix of coordination mechanisms that it deems most suitable to manage dependencies present in the task. Because distance changes the context of the task, different coordination mechanisms may be more or less effective. For example, geographic dispersion affects the nature and frequency of interaction within the team (Allen 1977). So, distributed teams will often need to purposely plan their meetings and communicate through electronic media. Thus, these teams may find it helpful to implement programs and other task

organization mechanisms (e.g., shared calendars, instant messengers) to overcome their reduced and less rich communication. At the same time, their lack of shared context and reduced level of interaction will make it more difficult for distributed teammates to develop team cognition. This suggests that:

H.4: *Task programming mechanisms will be more effective than communication mechanisms and shared knowledge of the task and team for coordination in large-scale, geographically distributed software development*

Research Methodology

To evaluate our hypotheses, we conducted a field study of coordination in geographically distributed software development teams at a major telecommunications firm. The field study included a survey of software teams in the company, supplemented with archival data on the software projects they completed. The software teams surveyed in this study produce software for digital telephony switching equipment. Digital telephony switching equipment represents the largest revenue product category for the company, and it employed approximately 4,000 developers at the time of the study. This switch product contains a very large and complex real-time software program, containing over 200 million lines of code and is composed of a few large sub-systems (e.g., common channel signaling). The software for different sub-systems is produced in different internal organizations that are in different geographical locations.

The software is updated with incremental releases (i.e., versions) that contain a number of new and/or updated features. A feature is a specific functionality added to, or improved in the software product (e.g., billing, call waiting, etc.), implemented with one or many software modifications. Software modifications are implemented via "MRs" or MRs. An MR is the basic unit of software work in the software organization we studied and it is similar in concept to a work order. No changes can be made to the software product without a formally approved MR, which can be issued for either new feature development or existing software repairs. Almost

anyone in the software organization can issue an MR, which is then formally reviewed by a Change Control Board. The Change Control Board approves, rejects, or commissions the MR for further study. If approved, the Change Control Board assigns a priority to the MR, which is subsequently scheduled for implementation based on the multiple priorities of all approved MRs pending development. Once an MR enters the work planning cycle, it is assigned a supervisor, development personnel, a budget and other resources. An MR contains one or many software changes called "Deltas". A Delta is the basic unit of software change in these organizations and it involves the addition, deletion and/or modification of one or many lines of code by a single developer on a single file.

Unit of Analysis: Programming Teams for MRs

Because of the size and complexity of large-scale software products, and the large number of people and groups involved in these tasks, the nature of coordination problems in this domain can vary widely depending on the level of integration involved (e.g., MR, feature, sub-system, etc.) and the software phase (e.g., design, coding, testing, etc.). To develop a better understanding of work processes and coordination challenges at the root of the software development process, the unit of analysis selected for this study consists of coding teams for MRs. The software-coding phase was selected because this phase employs the majority of the software development staff at this organization, and this phase had the most severe coordination problems. So, improvements in coordination in the coding of MRs can have a substantial impact on overall performance. The MR was selected as the unit of analysis because it is the basic unit of software work at this organization. In addition, the MR has a number of desirable characteristics for a unit of analysis, including: it is a formally approved unit of software work and has assigned priorities and resources; its implementation team is fairly small and well defined, thus facilitating the analysis; its scope of work is also well defined, and its task

dependencies are more easily identified; and most of the available software production data in archives are traceable to the MR.

Sample

Because software for different switch sub-systems is produced by different internal organizations, and to avoid confounds stemming from inter-organizational differences, this study surveyed software developers for a single sub-system, "Common Channel Signaling" (CCS). This sub-system contains over 25% of the switch's software. The survey was administered in two sites—England and the U.S.—where almost all of the code for this sub-system was produced. The sample selected for the study included all MRs in the Common Channel Signaling sub-system completed within the last three years in which two or more currently employed developers were involved. A three-year timeframe was selected based on recommendations from two senior software production researchers from this company and by an experienced switch software developer. Think-out-loud pre-testing of the survey instrument with two developers confirmed that the three-year timeframe was adequate for recall purposes. Furthermore, to facilitate recall, the web survey instrument contained links to the respective MR database records. The web survey was dynamically generated for each respondent with questions about one to three actual MRs in which the respondent contributed Deltas, and questions about each developer who collaborated in these MRs. The limitation of three MRs per respondent reduced the sample size substantially, but it was necessary to keep the survey completion time to 30 minutes, as requested by managers in the software organizations studied. A total of 113 participants were identified per the criteria discussed above, and 97 of them completed the survey, yielding an 85.8% response rate and 60 MRs, which was reduced to 54 MRs after discarding 6 incomplete entries. MR teams in the sample ranged from 2 to 7, with an average team size of 3.5.

Survey Instrument

The web survey was implemented so that important information about MRs and about the particular developers who worked on each MR was dynamically retrieved from a database, such that the web survey instrument was customized for each participant. Consequently, most survey items contained questions related to the particular MRs (in the sample) that the participant had worked on (e.g., "please rate the complexity of this MR") or about each team member who worked on the MR (e.g., "this team member possessed the necessary skills for this MR").[1] Survey items from prior the empirical research literature were utilized as much as possible. Items that were not available in prior research studies were developed jointly with two researchers from a software production research department from this company.

Study Variables

Most variables in this study come from responses to the dynamic web survey. Several quantitative variables that measure aspects of the code or MRs were obtained from the software configuration management system's database. Variables computed at the individual level were averaged to the team level. Analyses of intra-class correlation (ICC) (Kenny and LaVoie 1985) and within-team agreement R_{WG} (James, Demaree et al. 1984) statistics were computed for each variable to justify the aggregation of individual-level variables to the team level. All variables, except one independent variable, exhibited ANOVA p-values of 0.25 or less, and positive ICC's and R_{WG}, indicating that aggregating to the team level is appropriate. Each of the variables constructed for this study are explained below, starting with the dependent variable (i.e., coordination), followed by the main independent variables (i.e., SMM Team and SMM Task). Other variables of interest (i.e., task organization, team communication) are explained next,

[1] The questions for the items in the web-based survey are available upon request from the authors.

	Variable	Mean	Std. Dev.	1	2	3	4	5	6	7	8	9	10	11	12	13
1	Coordination Success	5.37	0.86													
2	Shared Knowledge of the Task	0.40	0.34	-0.10												
3	Shared Knowledge of the Team	0.49	0.29	-0.17	0.19											
4	Use of SW Configuration Mgt Syst	5.17	0.91	0.21	-0.06	-0.02										
5	Use of Software Tools	3.25	1.19	-0.20	0.15	0.39	-0.09									
6	General Communication	4.09	1.30	-0.15	-0.06	0.14	0.42	-0.06								
7	Telephone Communication	1.88	1.08	-0.07	-0.15	0.07	-0.04	0.25	0.17							
8	Distance Between Team Members	1.91	1.00	0.07	0.10	-0.07	-0.50	0.16	-0.52	0.15						
9	Dependency	4.31	1.26	-0.62	0.03	0.31	-0.05	0.08	0.12	0.20	0.00					
10	Lines of Code Modified	2,920.89	5,491.90	-0.22	-0.14	0.17	-0.25	0.28	-0.05	0.27	0.11	0.39				
11	Team Size	3.20	1.31	-0.35	0.24	0.16	-0.27	0.43	-0.12	0.36	0.18	0.37	0.31			
12	MR Start Date	558.96	283.58	-0.12	-0.08	0.16	0.02	0.15	-0.19	0.01	0.00	-0.05	-0.20	0.08		
13	New Development (0,1)	0.67	0.48	-0.41	0.19	0.20	-0.14	0.26	0.14	0.12	0.05	0.17	0.24	0.32	-0.12	
14	MR Priority (1-High, 4 Low)	3.02	0.69	-0.25	0.18	0.12	-0.04	0.20	0.11	0.17	0.02	0.18	0.33	0.29	0.10	0.48

p<0.001 for r>0.45, p<0.01 for r>0.31, p<0.05 for r>0.27

Table 1
Descriptive Statistics and Correlation Matrix

followed by control variables. Table 1 shows descriptive statistics and the correlation matrix for all of the variables in the analysis.

Coordination Success

All coordination survey items were analyzed using factor analysis. Items measured technical (e.g, "We had a lot of integration problems associated with this MR"), temporal (e.g., "We often had to do re-planning because of missed delivery dates and delays associated with this MR") and process (e.g., "We had many problems due to priority conflicts/confusions with this MR") coordination success separately. A three-factor solution was first obtained with high factor loadings roughly coinciding with the items selected for each of these types of coordination, suggesting that there are three different types of coordination success. However, two of the resulting factors were correlated and the third factor only contained a single item, so all eleven coordination items were analyzed for reliability. One item was removed because its removal improved the reliability of the measure. Consequently, the coordination success variable used for the study is the average of the other ten items (Cronbach-α=0.919). The fact that technical and process coordination factors and items were highly correlated was not surprising because if the

software process is not coordinated, chances are that many technical aspects will not be well coordinated either.

Well-coordinated teams working in tasks with substantial dependencies (e.g., large-scale software development) are more likely to have higher levels of performance, other things being equal (Thompson 1967; Van de Ven, Delbecq et al. 1976). Reduced software development time is an important measure of performance in the telecommunications software domain because it reduces time to market, which is very important for competitiveness in this industry. Partial correlation statistics showed that the coordination variable constructed in this study was significantly and negatively correlated to software development time (r=-0.39, p=0.010), thus providing some assurance of its convergent validity.

Shared Knowledge of the Task (SK Task) and of the Team (SK Team)

A recent comprehensive methodological review on team knowledge measurement suggested the use of similarity metrics to measure shared knowledge using inter-rater reliability or within-team agreement scores (Cooke, Salas et al. 2000). Consistent with prior studies, we use within-team agreement scores (James, Demaree et al. 1984) to measure knowledge similarity among team members (Levesque, Wilson et al. 2001; Rentsch and Klimoski 2001). While there has been some disagreement about whether to use the within-team agreement statistic (R_{WG}) as a reliability measure (Schimidt and Hunter 1989), R_{WG} is a popular measure of within-team agreement (Kenny and LaVoie 1985; James, Demaree et al. 1993). Because shared knowledge is about knowledge similarity (i.e., agreement) and not about reliability, R_{WG} is an appropriate measure.

Consistent with methods used in other studies (Mathieu, Goodwin et al. 2000; Levesque, Wilson et al. 2001; Rentsch and Klimoski 2001), we asked participants a number of task-relevant questions related to the particular MRs they worked on (e.g., "Please rate the

complexity of this MR"), and then computed the R_{WG} for each question asked. The shared knowledge of the task variable was computed as the average R_{WG} for all questions. The response scales for this construct were implemented with a neutral mid point and were worded to facilitate actual interpretation of scales (e.g., how much work on this MR involved repair work? Scale options: none, very little, a small amount, about 50%, a large amount, most of it, all of it). Discussions with an experienced developer in this organization and feedback from survey pre-tests suggested that developers could make fairly accurate interpretations of the scales used.

Similarly, we asked participants to rate a number of items (e.g., "knowledge of specifics about the MR was very high") about each team member who worked on their MRs and then computed the R_{WG} for each question asked. The shared knowledge of the team variable was computed as the average R_{WG} for all questions. Partial correlation statistics showed that the shared mental model of the task measure constructed in this study was significantly and positively correlated with familiarity with the same files and modules ($r=0.30$, $p=0.036$), thus providing some assurance of the convergent validity of the measure for shared mental model of the task. Team members who have worked previously on the same projects and products are more likely to have shared knowledge of each other than those who have not. Partial correlation statistics showed that the shared mental model of the team measure constructed in this study was moderately and positively correlated with prior familiarity with the same MRs ($r=0.29$, $p=0.052$), thus providing some assurance of the convergent validity of the measure for shared mental model of the team.

Task Programming: Use of Software Configuration Management System

Large-scale software development employs configuration management systems because these tools enable multiple developers to work on similar parts of the code simultaneously, and then merge their changes into an integrated module without interfering with each other. The

configuration management system also keeps track of different versions and variants that make up different releases of the software product. In addition, it contains a number of features to help developers manage the general software process (e.g., developer roles, workflow management) and record problems (e.g., error logs, developer comments) (Grinter 2000).

All developers in this software organization are required to use the configuration management system for standard purposes (e.g., checking out/in files, logging Deltas) to ensure that parallel changes to, and multiple versions/variants of the software are adequately managed. However, some participants suggested that many developers used the system to report errors, communicate concerns, and describe technical details, among other things. Consequently, the configuration management system can be viewed as both a task organization tool that helps manage workflow, and as a communication tool that helps developers communicate technical issues. Because the use of this tool is mandatory for workflow purposes, we expect most of its use variance to come from its use for communication purposes.

The use of the configuration management system variable was constructed by averaging two survey items—use of the configuration management system and use of technical web sites. Factor analysis of task organization variables found that these two variables loaded highly on a single factor and that these variables were correlated ($r=0.319$, $p=0.003$). This is not unexpected because the item about use of technical web sites made explicit reference to a site that provides a web interface to configuration management system data, with links to most of the information available in its database.

Task Programming: Use of Software Tools

This variable was measured with a single survey item (i.e., "we used software tools extensively for this MR"). This is the only independent variable in which the ICC and R_{WG} were negative, suggesting that the use of software tools (other than the configuration management

system) is more of an individual preference than a group preference. Nevertheless, this variable was included in the model to control for the aggregate level of tool usage in the team and because it co-varies with one of the independent variables of interest (SK Team, r=0.387, p=0.004) suggesting that teams with more aggregate use of software tools have higher SK Team values. Consequently, this variable was included in the model because its omission would have biased the SK Team coefficient (Greene 1997, p. 401).

Team Communication: General Communication

Communication survey items were collected at the MR level and at the dyad level. MR-level communication items were used to study differences in communication modes (e.g., formal meetings, spontaneous meetings, etc.) within the team (e.g., "we had frequent formal review meetings for this MR"), but results from factor analysis yielded a single factor solution explaining 58.6% of the variance, showing no clear distinction between communication modes. Dyad-level data items were included in the survey to measure the communication frequency between every teammate pair in the MR team through different media (e.g., telephone, face-to-face). Dyad-level measures are better than team level measures for communication frequency because they help reduce common method variance bias (i.e., the dependent variable, coordination, is measured at the MR level). Results from factor analysis of the dyad-level communication items yielded a 2-factor solution explaining 85.6% of the variance. Three of the four variables (i.e., e-mail communication, face-to-face communication and overall communication) loaded highly on the first factor explaining 60.9% of the variance, suggesting that there is no distinction in communication frequency across media (except for telephone) for these technical teams. This is probably the case because the majority of their work is done using computers, thus making e-mail communication as frequent as face-to-face communication.

Therefore, the three items were averaged into a total communication variable (Cronbach-α=0.844).

Team Communication: Telephone Communication

Interestingly, telephone communication was the only dyad-level survey item that did not load highly on the first factor. It loaded on the second factor with an eigenvalue of approximately 1, explaining an additional 24.7% of the variance. This suggests that the technical teams surveyed use telephone communication in special circumstances, which differ from the general communication pattern. Consequently, telephone communication was included in the model as a single-item variable.

Control Variable: Distance Between Team Members

We controlled for differences across teams in the physical distance between team members. It is important to do this because the effects of geographic dispersion on communication can begin to take effect sharply as the physical distance between collaborators exceeds a few hundred feet (Allen 1977). We measured the distance between each dyad (pair of members) on a team using a 7-point Likert scale (1=same room; 2=same floor; 3=same building; 4=same city; 5= same country; 6=same continent; 7= different continent). We then obtained an average distance between team members for each team by averaging the Likert scale ratings for all dyads in the team.

Control Variable: Dependency

We control for differences in the level of dependencies across MRs. The extent to which a task has tightly coupled dependencies will influence the extent to which the task needs to be coordinated (Thompson 1967; Van de Ven, Delbecq et al. 1976). In fact, results from other studies suggest that software work is often organized so that dependencies between the internal organizations involved are minimized, thus reducing coordination problems (Herbsleb and

Grinter 1999). Different MRs and team work styles may involve varying levels of dependencies, which can not only affect the need for coordination, but also the respondent's perception of whether the MR was coordinated or not. This item was measured by asking respondents two questions about whether their work was dependent on others' work and vice-versa. The measure was computed as the combined average of these two items for all MR team members.

Control Variable: Lines of Software Code Modified

Larger software modifications are more likely to encounter more coordination problems than smaller ones. Furthermore, larger MRs are inherently more complex, which may influence survey participants' perceptions of how coordinated was the development of a MR, therefore the need to control for MR size. This variable was computed using data from the configuration management system. The system records the number of lines of software code added to and deleted from each Delta of each MR. When a line is modified, the system records it as the deletion of one line and the addition of another one. This variable was computed as the sum of lines deleted and added in all the Deltas for the MR.

Control Variable: Team Size

As the size of a software team increases, the amount of software development work that needs to be done remains the same, but the coordination overhead of the software project increases (Brooks 1995). This can be intuitively demonstrated by the fact that a team of n developers has $n(n^2-1)/2$ possible communication lines and dependency links. Brooks argues that when a team of *n* software developers adds an additional developer, it has two effects: (1) it increases the developer resource pool by a fraction of *1/n*, which should help reduce development time; (2) but it also creates *n* new communication lines and dependency links that the new developer will have with each of the *n* existing developers. Most of these links need some maintenance and management, thus making coordination more difficult. Therefore, it is

important to control for team size. The variable used for this study comes from a survey item that asks participants to enter the number of developers involved in the MR. The variable used for this study was constructed as the average MR team size entered by survey respondents for that MR. The team size variable is highly correlated with the actual number of people who recorded Deltas for that MR in the configuration management system ($r=0.951$, $p<0.001$), thus providing good assurance of its convergent validity.

Control Variable: MR Start Date

To control for recall factors, MR age, and other time-related factors (e.g., technology changes, management changes, etc.), the MR start date variable was included in the model. This variable was computed as the date when the MR started relative to the earliest MR in the sample (start date = 0), measured in days.

Control Variable: New Development MR (0,1)

MRs are opened to either develop software for new features or to maintain existing software. New development MRs and maintenance MRs are different types of software tasks. Maintenance MRs include those opened to correct errors, improve performance, or adapt the product to changed conditions (ANSI/IEEE 1983). While new development MRs have uncertainties associated with the requirements of a new product, maintenance MRs have the added complexity of needing to understand the existing code before modifications can be made (Banker, Dattar et al. 1993; Kemerer 1995; Banker, Davis et al. 1998). The coordination challenges may vary between new and maintenance MRs, therefore, the type of MR needs to be controlled for. New development MRs in this domain are associated with the implementation of new features that add or improve functionality to an existing product, which contain dependencies with designers and customer requirements groups. Consequently, it is anticipated that MRs for new development will be perceived by respondents to require more coordination

than for software maintenance. This variable was constructed from information available in the configuration management system' database as a binary variable, equal to 1 for MRs involving new feature development and equal to 0 for MRs involving software maintenance.

Control Variable: Priority (1=highest, 4=lowest)

MRs approved for implementation are assigned 1 of 4 priority levels by the Change Control Board based on their impact on clients. Priorities determine how quickly MRs need to be implemented. A priority of 1 means that a fault has or could cause a serious service interruption. A priority of 2 is one assigned to a major potential system failure with potentially serious service impact. A priority of 3 is one in which there is a minor software recoverable problem that can result in inconveniences to a user or office. A priority of 4 is generally assigned to nuisance low-level problems with little effect on a user or office. Priorities for new development are also assigned based on these guidelines and other market factors. Priorities can affect coordination, particularly when emergency MRs are implemented in "crisis" mode. Consequently, we need to control for MR priority.

Analysis and Results

Initial results obtained using ordinary least squares (OLS) regression suggested the presence of heteroscedasiticy. Thus, weighted least squares (WLS) regression with a general weighting procedure was used to analyze the data as WLS produces more efficient estimators in the presence of non-spherical residuals (Kennedy 1992; Greene 1997). Results from the WLS regression analysis are shown in Table 2 and illustrated in Figure 1. A White test for heteroskedasticity revealed no further problems. The model was also inspected for multicollinearity using the condition index and variance inflation factors. Although the condition index was 36.74, suggesting moderate levels of multicollineary in the model as a whole, the

variance inflation factors for the estimated coefficients are well below 5 indicating no problems with multicollinearity between the individual regressors.

Variable	Standard Coefficient	P-Value	VIF
Use of Config Mgt Syst	0.359	**0.003**	1.562
Shared Knowledge of the Task	-0.008	0.726	1.411
Shared Knowledge of the Team	0.084	**0.041**	1.517
Use of Software Tools	-0.060	0.324	1.635
General Communication	-0.099	0.112	1.984
Telephone Communication	0.022	0.554	1.490
Distance Between Team Members	0.063	0.125	1.806
Dependency	-0.441	**0.000**	1.524
Lines of Code Modified	0.033	0.143	1.872
Team Size	0.040	0.459	1.942
MR Start Date	-0.095	**0.022**	1.434
New Development MR (0,1)	-0.103	**0.009**	1.533
Priority (1-High, 4 Low)	0.062	0.641	1.624

Table 2: Regression Analysis Results

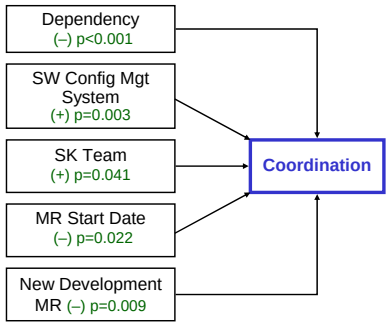

Figure 1: Survey Study Results

H1 predicted that use of task programming mechanisms would increase coordination effectiveness in large-scale, geographically distributed software development. Our empirical results provide mixed support for this hypothesis, as the use of a software configuration management is positively and significantly associated with coordination effectiveness system (β= 0.359, p=0.003) while use of other software tools is not. H2 posited that use of communication mechanisms would increase coordination effectiveness in large-scale,

geographically distributed software development. This hypothesis was not supported, as the coefficients for telephone communication and general communication are not significant. H3 hypothesized that shared knowledge of the team and task would promote coordination effectiveness in large-scale, geographically distributed software development. This hypothesis is partially supported as SK Team had a significant and positive effect on team coordination effectiveness (β=0.084, $p<0.041$). However, SK Task was not significantly associated with team coordination. Our final hypothesis (H4) posited that geographically distributed developers working on large-scale software development projects will rely more on task programming mechanisms than on mechanisms leveraging communication or shared knowledge. Our results in Table 2 suggest support for this hypothesis as the standardized coefficients reveal that the software configuration management tool (a task programming mechanism) has a significantly greater effect on coordination than either shared knowledge mechanisms (more than three times the positive effect of shared knowledge of the team on coordination effectiveness) or team communication mechanisms (which are not significantly related to coordination effectiveness).

In terms of the control variables, the strongest predictor of coordination was member dependency (β= –0.441, $p<0.001$). Consistent with prior studies (Van de Ven, Delbecq et al. 1976; Herbsleb and Grinter 1999), these results provides empirical evidence that MRs in which team members work more independently are perceived to be better coordinated. This confirms the concerns of software engineers about the importance of de-coupling software tasks in such a way that dependencies between groups that have more coordination overhead (e.g., across locations, across organizations, etc.) are minimized. The MR start date coefficient was negative and significant (β= –0.095, $p=0.022$) suggesting that more recent MRs are perceived to be less coordinated than older ones. This result is not surprising because the telephony switch software studied is developed using the incremental method by adding new features or improving existing

ones. Consequently, as the product gets older, larger and more complex, and as experienced software engineers who were familiar with the software code leave the organization, respondents are more likely to find it more difficult to coordinate. Also, coordination problems with more recent MRs are more likely to be salient in people's memories, which is one of the reasons why the start date variable was included—i.e., to control for recall factors. Finally, the only other control variable that was significant was MR type, suggesting that new development MRs have lower coordination effectiveness than maintenance MRs (β= –0.103, $p=0.009$), as expected.

Discussion

One interesting finding is that shared knowledge of the team facilitated coordination, while shared knowledge of the task did not. At the same time, the effect of the use of the configuration management system was significant. These results combined suggest that in large-scale software development carried out in geographically dispersed contexts it is important to know team members, which is consistent with arguments from transactive memory research (Wegner 1995). The fact that shared knowledge of the task is not significant, but the use of the configuration management system is significant suggests that when software teams have effective collaboration tools that are tailor made to the tasks, and that facilitate the exchange, discussion and retrieval of key task information and knowledge, then having shared task knowledge among members is not as important, because the knowledge can be easily found. Our results imply that global software development organizations can benefit substantially from promoting the use of practices and collaboration tools that strengthen shared knowledge among team members.

The results for the use of the software configuration management tool are also intriguing. As we noted earlier, developers were required to use the tool, so any variation in use of the tool can be attributed to its use for communication. In fact, a review of the text recorded in the

configuration management system for the MRs in the study sample revealed substantial variance in the amount of text entered in these records: some MRs have a substantial amount of descriptive detail, while others have very little. For example, the shortest entry only contained one comment line saying “work on editor complete/discontinued”, and most of the mandatory description fields were empty. In contrast, the largest entry had a substantial amount of detailed information, including: feature overview, discussions of priorities, problem description, description of work needed, software code examples, references to formal documents and manuals, general requirements, design issues, data files affected, testing issues, and customer documentation requirements. This suggests that the developers used the tool not only as a task programming mechanism but also as a communication mechanism. In a geographically distributed context, it may be more efficient for developers to use tools not only to coordinate technical details of their work but also to communicate with each other about the work.

This research makes several important contributions to both the research literature and the practitioner community. Most of the prior research on coordination has not investigated the joint effect of various types of coordination mechanisms in geographically distributed work. This research uniquely combines two views of coordination – the traditional view from the classic organizational literature (i.e., task programming and communication), and the more recent view from team cognition research. It is also the first study that has tested the effect of shared knowledge on coordination in geographically distributed software development (i.e., an asynchronous task) in a real organizational context. Finally, even though there are many collaboration tools that support distributed and asynchronous collaboration, we know very little about the effectiveness of these tools. Future research ensuing from the findings of this study can help identify key features for the next generation of collaboration tools based on shared knowledge, not just on task programming and communication.

References

Allen, T. (1977). Managing the Flow of Technology. Cambridge, MA, MIT Press.

ANSI/IEEE (1983). Standard 729, IEEE Standard Glossary of Software Engineering Terminology.

Banker, R., Dattar, S. and Kemerer, C. (1993). "Sofware Complexity and Software Maintenance Costs." Communications of the ACM 36(11): 81-94.

Banker, R., Davis, G. and Slaughter, S. A. (1998). "Software Development Practices, Software Complexity, and Software Maintenance Performance: A Field Study." Management Science 44(4): 433-450.

Brooks, F. (1995). The Mythical Man-Month: Essays on Software Engineering, A. Wesley.

Cannon-Bowers, J. A., Salas, E. and Converse, S. (1993). Shared Mental Models in Expert Team Decision-Making. Individual and Group Decision-Making: Current Issues. J. Castellan, LEA Publishers**:** 221-246.

Carmel, E. (1999). Global Software Teams. Upper Saddle River, NJ, Prentice Hall.

Cooke, N. J., Salas, E., Cannon-Bowers, J. A. and Stout, R. J. (2000). "Measuring Team Knowledge." Human Factors 42(1): 151-173.

Crowston, K. and Kammerer, E. E. (1998). "Coordination and Collective Mind in Software Requirements Development." IBM Systems Journal 37(2): 227-245.

Curtis, B., Krasner, H. and Iscoe, N. (1988). "A Field Study of the Software Design Process for Large Systems." Communications of the ACM 31(11): 1268-1286.

Faraj, S. and Sproull, L. (2000). "Coordinating Expertise in Software Development Teams." Management Science 46(12): 1554-1568.

Greene, W. (1997). Econometric Analysis, Prentice Hall.

Grinter, R. E. (2000). "Workflow Systems: Occasions for Success and Failure." Computer Supported Cooperative Work 9: 189-214.

Hackman, R. (1987). The Design of Work Teams. Handbook of Organizational Behavior. J. Lorsch, Prentice-Hall.

Herbsleb, J., Mockus, A., Finholt, T. and Grinter, R. E. (2001). An Empirical Study of Global Software Development: Distance and Speed. 23rd. International Conference on Software Engineering (ICSE), Toronto, Canada, IEEE Computer Society Press.

Herbsleb, J. D. and Grinter, R. E. (1999). "Architectures, Coordination, and Distance: Conway's Law and Beyond." IEEE Software 16(5): 63-70.

Herbsleb, J. D., Mockus, A., Finholt, T. and Grinter, R. E. (2000). Distance, Dependencies and Delay in a Global Collaboration. 2000 ACM Conference on Computer Supported Collaboartive Work, Philadelphia, Pennsylvania, ACM Press.

James, L. R., Demaree, R. G. and Wolf, G. (1984). "Estimating Within-Group Interrater Reliability With and Without Response Bias." Journal of Applied Psychology 69(1): 85-98.

James, L. R., Demaree, R. G. and Wolf, G. (1993). "Rwg: An Assessment of Within-Group Interrater Agreement." Journal of Applied Psychology 78(2): 306-309.

Kemerer, C. (1995). "Software Complexity and Software Maintenance: A Survey of Empirical Research." Annals of Software Engineering(1): 1-22.

Kennedy, P. (1992). A Guide to Econometrics, MIT Press.

Kenny, D. A. and LaVoie, L. (1985). "Interpersonal Relations and Group Processes." Journal of Personality and Social Psychology 48(2): 339-348.

Klimoski, R. J. and Mohammed, S. (1994). "Team Mental Model: Construct or Methaphor." Journal of Management 20(2): 403-437.

Kraiger, K. and Wenzel, L. (1997). Conceptual Development and Empirical Evaluation of Measures of Shared Mental Models as Indicators of Team Effectiveness. Team Performance Assessment and Measurement. M. Brannick, E. Salas and C. Prince, LEA Publishers: 63-84.

Kraut, R. E. and Streeter, L. A. (1995). "Coordination in Software Development." Communications of the ACM 38(3): 69-81.

Levesque, L. L., Wilson, J. M. and Wholey, D. R. (2001). "Cognitive Divergence and Shared Mental Models in Software Development Project Teams." Journal of Organizational Behavior 22(2): 135-144.

Liang, D., Moreland, R. and Argote, L. (1995). "Group Versus Individual Training and Group Performance: The Mediating Role of Transactive Memory." Personality and Social Psychology Bulletin 21.

Malone, T. and Crowston, K. (1990). What is Coordination Theory and How Can it Help Design Cooperative Work Systems. Computer Supported Collaborative Work, Los Angeles, CA, ACM Press.

Malone, T. and Crowston, K. (1994). "The Interdisciplinary Study of Coordination." ACM Computing Surveys 26(1): 87-119.

March, J. and Simon, H. (1958). Organizations. New York, John Wiley and Sons.

Mathieu, J., Goodwin, G. F., Heffner, T. S., Salas, E. and Cannon-Bowers, J. A. (2000). "The Influence of Shared Mental Models on Team Process and Performance." Journal of Applied Psychology 85(2): 273-283.

Rentsch, J. R. and Hall, R. J. (1994). "Members of Great Teams Think Alike: A Model of the Effectiveness and Schema Similarity Among Team Memebers." Advances in Interdisciplinary Studies of Work Teams 1: 223-261.

Rentsch, J. R. and Klimoski, R. J. (2001). "Why do Great Minds Think Alike? Antecedents of Team Member Schema Agreement." Journal of Organizational Behavior 22(2): 107-120.

Schimidt, F. L. and Hunter, J. E. (1989). "Interrater Reliability Coefficients Cannot Be Computed When Only One Stimulus is Rated." Journal of Applied Psychology 74: 368-370.

Sproull, L. and Kiesler, S. (1991). Connections: New Ways of Working in the Networked Organization, MIT Press.

Thompson, J. (1967). Organizations in Action, McGraw-Hill.

Van de Ven, A. H., Delbecq, L. A. and Koenig, R. J. (1976). "Determinants of Coordination Modes Within Organizations." American Sociological Review 41(April): 322-338.

Walz, D. B., Elam, J. J. and Curtis, B. (1993). "Inside a Software Design Team: Knowledge Acquisition, Sharing, and Integration." Communications of the ACM 36(10): 63-77.

Wegner, D. (1995). "A Computer Network Model of Human Transactive Memory." Social Cognition 12(3).

Weick, K. and Roberts, K. (1993). "Collective Mind in Organizations: Heedful Interrelating on Flight Decks." Administrative Science Quarterly 38(3): 357-381.

Wittenbaum, G. M. and Stasser, G. (1996). Management of Information in Small Groups. What's Social about Social Cognition? J. L. Nye and A. M. Brower, Sage Publications: 3.