Coordination Breakdowns and their Impact on Development Productivity and Software Failures

Marcelo Cataldo and James D. Herbsleb

March, 2010
CMU-ISR-10-104

Institute for Software Research
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213-3890

Abstract

The success of software development projects depends on carefully and effectively coordinating the effort of many individuals across the multiple stages of the development process. In software engineering, modularization is the traditional technique intended to reduce the interdependencies among modules that constitute a system. Reducing technical dependencies, the theory argues, results in a reduction of work dependencies between teams developing interdependent modules. Organizational researchers have proposed similar theoretical arguments. Although such research streams have been quite influential, they have taken a coarse-grain and static view of the problem of coordination in engineering activities. This paper proposes a new perspective on coordination where fine-grain and evolving dependencies are front and central. Our empirical analyses demonstrate that considering dependencies at a fine-grain level of analysis provides us deeper insight to the relationship between technical and work dependencies. Moreover, our examination of two large scale projects from two distinct companies showed that (a) logical dependencies among software entities are significantly more important in terms of coordinating development work compared to syntactic dependencies and (b) satisfying coordination needs arising from those logical software dependencies with appropriate coordinating actions results in significant improvements on development productivity as well as a significant reduction in the failure proneness of the software systems.

Keywords: metrics / measurement, productivity, organizational management and coordination, quality analysis and evaluation.

I. INTRODUCTION

Over the past decades, software-intensive systems have become increasingly pervasive in our lives, and their complexity has increased drastically (Ebert & Jones, 2009). Software development organizations are also changing. Factors such as project scale, access to talent, acquisitions, and the need to reduce the time-to-market have fueled an increase in distribution of the development work and a corresponding increase in organizational complexity (Herbsleb & Moitra, 2001; Karolak, 1998; Northrop et al, 2006). Coordination – long recognized as one of the fundamental problems of software engineering (Curtis et al, 1988; Kraut & Streeter, 1995) – has become ever more challenging. This has led to a growing body of empirical work on coordination in software development (e.g. Cataldo et al, 2007, 2009; de Souza et al, 2004; Grinter et al, 1999; Herbsleb et al, 2000; Herbsleb & Mockus, 2003).

Coordination is a topic that has received significant research attention in both modular product design and organizational theory. The key idea across those bodies of research is "nearly decomposable" systems (Simon, 1962). On the technical dimension, the work on modular product design has examined extensively the role of interdependencies among components of a product and has proposed approaches to minimize those dependencies (Baldwin & Clark, 2000; Eppinger et al, 1994; Parnas, 1972; Sullivan et al, 2001). A key assumption in this line of work is that minimizing technical dependencies among product components will result in a "modular work structure" (Baldwin & Clark, 2000; Parnas, 1972). The organizational theory literature has developed similar theoretical arguments from the perspective of organizing the work around loosely-coupled entities such as teams or departments (Galbraith, 1973; Malone & Crowston, 1994; March & Simon, 1958; Staudenmayer, 1997; Thompson, 1967).

Although both research streams have been quite influential, those theoretical perspectives have important limitations. A modular strategy is vulnerable to unanticipated "cross-cutting" product features, as they require coordinated changes to multiple modules (e.g. Kiczales & Mezini, 2005). Moreover, modular structures as well as traditional organizational mechanism to coordination are not suitable for environments with volatile dependencies (e.g. Cataldo et al, 2006; Faraj & Xiao, 2006; Gittel, 2002; Staudenmayer, 1997). In sum, the dominant theoretical perspectives have taken a coarse-grain and static view of the problem of coordination in engineering activities and they fail to fully recognize the complex and dynamic reality of software development projects. This paper proposes a new perspective on coordination where fine-grain and evolving dependencies are front and central. Our empirical analyses demonstrate that considering dependencies at a fine-grain level of analysis provides us deeper insight to the relationship between technical and work dependencies. Moreover, our examination of two large scale projects from two distinct companies showed that matching fine-grained coordination needs with appropriate coordinating actions results in significant improvements productivity as well as a significant reduction in defects.

The remaining of the paper is organized as follows. We first examine the limitations of current theoretical views of coordination from both a technical and an organizational perspective. Second, we propose a new framework to examine the relationship between the structure of software systems and the corresponding organizational structures. Third, we describe our research setting followed by the empirical analyses. We conclude with a discussion of the implication of our results and future research directions

II. Dependencies in Software Development: The Traditional Perspective

Traditional approaches to coordination are based on a reductionist view which argues that complexity, technical or organizational, can be managed by dividing a system or task into smaller and more manageable units (March & Simon, 1958; Simon, 1962; von Hippel, 1990). In a technical context, the work is often characterized as choosing the design parameters for a system (Baldwin & Clark, 2000; Browning, 2001). Such decisions are often highly inter-related, of course, and a modular approach creates system components that are, in effect, "bundles" of design decisions. Good architectural design bundles decisions in such a way that there are relatively few decisions that affect multiple components. These decisions that do affect multiple components, often called architectural decisions (Shaw & Garlan, 1996) are made first, and establish the stable "design rules" that determine how components will interact (Baldwin & Clark, 2000). In this strategy, the remainder of the engineering work should consist of designing the components within the constraints established by the design rules. This permits different organizational units (Conway, 1968; Parnas, 1972; Eppinger et al, 1994; Sullivan et al, 2001), or even different organizations (Baldwin and Clark, 2000) to work in parallel on different components. Conway (1968) proposed that the component structure and organizational structure stand in a homomorphic relation, in that more than one component can be assigned to a team, but a component must be assigned to a single team. A similar argument has been proposed in the product development literature. Baldwin and Clark (2000, page 90) argued that modularization makes complexity manageable, enables parallel work and tolerates uncertainty.

It is, of course, recognized that components may be merely "nearly" rather than strictly decomposable (Simon, 1962). On the organizational dimension, researchers have proposed numerous approaches to manage the work dependencies that might remain across organizational enti-

ties such as individual, teams or departments. In the traditional organizational theory, March and Simon (1958) argued that schedules and feedback mechanisms are required when interdependence is unavoidable. Thompson (1967) extended March and Simon's work by matching three mechanisms: standardization, plan, and mutual adjustment, to stylized categorizations of dependencies such as pooled, sequential, and reciprocal. Galbraith (1973) argued that low levels of interdependency can be managed by traditional mechanisms such as rules and programs. However, as the level of interdependency increases additional mechanisms are required such as slack resources and lateral communication (Galbraith, 1973). Mintzberg (1979) took an organizational-level perspective and argued that specific coordination mechanisms are properties of particular kinds of organizations and environments. Malone and Crowston (1994) developed a typology of coordination problems to catalog coordination mechanisms that address specific types of interdependencies.

The predominant theoretical perspectives on dependencies and coordination have important limitations that diminish their applicability in the software engineering context and limit their ability to address the challenges of identifying and managing dependencies. First, the modularization argument assumes a single modular structure is sufficient for all purposes. Yet, existing software modularization approaches consider only a subset of the technical dependencies, typically syntactic relationships (Garcia et al, 2007). Yet other concerns, such as semantics, system performance, security, and reliability each create different webs of dependencies that cannot all be captured in a single structure (Clements, et al, 2002). Moreover, modularization and organizational theories assume a predictable and static technical structure that is mirrored by a static organizational structure (Henderson & Clark, 1990; Faraj & Xiao, 2006; Staudenmayer, 1997). However, it is widely accepted among software engineering researchers and practitioners that the

requirements of the system become known gradually over time, and requirements change as time progresses (Curtis et al, 1988; Leffingwell & Widrig, 2003). In some cases the changes in the requirements result in minor alterations of specific development tasks. In other cases, new features have to be added or features under development are eliminated. Such changes disrupt component interfaces, i.e., design rules, by establishing new dependencies among the various parts of the system, modifying existing ones, or even eliminating dependencies.

The effects of such changes on task dependencies can be quite subtle and difficult to identify a priori (Henderson & Clark, 1990; Sosa et al, 2004). For example, Cataldo et al (2007) presented case studies where even simple interfaces between modules developed by remote teams create coordination breakdown and integration problems. In a field study of a large software project, de Souza (2005) observed that interfaces tended to change often and their design details tended to be incomplete, leading to serious integration problems. Failure to discover the changes in coordination needs might have a profound impact on the quality of the product (Curtis et al, 1988), on productivity (Herbsleb & Mockus, 2003) and even on the projects' overall design (Bass et al, 2007).

In sum, traditional views of coordination are based on the notion that a single modular structure is adequate for all purposes, and assume a static organization and a static and predictable product structure. Research has shown that, for a wide variety of reasons, these assumptions frequently fail. Software development tasks are embedded in an evolving network of task dependencies. The coarse-grain and idealized approaches suggested by the modularization and organization theory literatures are not adequate to identify and manage such a dynamic web of interdependencies. A finer-grain view of coordination that seeks to identify and continuously monitor evolving coordination needs has the potential to provide a better framework.

III. SOCIO-TECHNICAL CONGRUENCE: A FINE-GRAIN VIEW OF DEPENDENCIES

In this section, we present a framework to determine the coordination requirements among developers. The objective of the framework is to provide a fine-grain level of analysis of coordination that does not assume a single structure or a static architecture. To do so, we analyze files of source code, which represent "bundles" of design decisions that are far smaller than components (in our data, 1-2.5 orders of magnitude smaller), and compute task dependencies from data automatically collected by standard software development tools. We also propose a measure of "fit" between these work dependencies and the coordination activities performed by the software developers.

Two lines of work are particularly relevant in this context. The concept of "fit" from organizational literature refers to the match between a particular organizational design and the organization's ability to carry out a task (Burton & Obel, 1998; Carley & Ren, 2001; Levchuck et al, 2004). Research on dynamic analysis of social networks provides an innovative approach, called the meta-matrix, to examine the dynamic co-evolution of relationships among multiple types of entities such as resources, tasks, and individuals (Carley, 2002; Krackhardt & Carley, 1998). The concept of socio-technical congruence presented in this paper draws on these two lines of research to provide three important contributions to the literature. First, the socio-technical congruence framework presented here provides a fine-grain level of analysis that we argue is essential. Secondly, the congruence measure facilitates assessing the role of coordination activities, and allows us to identify both the impact of the modular strategy and the impact of using lightweight communication media to manage task dependencies that violate the modularity assumptions. Finally, we examine how task dependencies and coordination evolve over time, finding that the

modular strategy is fairly effective early in a project, but breaks down fairly dramatically as a project progresses.

Socio-technical congruence is generally defined as the match between the coordination requirements established by the dependencies among tasks and the actual coordination activities carried out by the workers. In other words, the concept of congruence has two components, coordination needs and coordination activities. In order to determine the coordination needs, we need to represent two sets of relationships: which individuals are working on which development tasks and the dependencies among tasks. In the framework proposed in this study, assignments of individuals to particular work items is represented by a people by task matrix where a one in cell *ij* indicates that worker *i* is assigned to task *j*. We will refer to this matrix as *Task Assignments* (T_A). The set of dependencies among tasks can be represented as a square matrix where a cell *ij* (or cell *ji*) indicates that task *i* and task *j* are interdependent. We will refer to this matrix as *Task Dependencies* (T_D). Now, if the *Task Assignment* and *Task Dependencies* matrices are multiplied, a people by task matrix is obtained that represents the set of tasks a particular worker should be aware of, given the tasks the person is responsible for and the dependencies among tasks. Finally, a representation of the coordination requirements among the different workers is obtained by multiplying the product of the *Task Assignment* and *Task Dependencies* matrices by the transpose of the *Task Assignment* matrix. This product results in a people by people matrix where a cell *ij* (or cell *ji*) indicates the extent to which person *i* works on tasks that share dependencies with the tasks worked on by person *j*. In other words, the resulting matrix represents the *Coordination Requirements* or the extent to which each pair of people needs to coordinate their work. More formally, the *Coordination Requirements* matrix is determined by the following product:

$$C_R = T_A * T_D * T_A^T \qquad (1)$$

where, T_A is the *Task Assignments* matrix, T_D is the *Task Dependencies* matrix and T_A^T is the transpose of the *Task Assignments* matrix.

So far, we have discussed coordination requirements – we now turn our attention to coordination activities. *Actual Coordination* (C_A) is represented by a square, people by people matrix, where each entry in cell *ij* (or cell *ji*) represents the extent to which person *i* engaged in a particular kind of coordination activity with person *j*.

Socio-technical congruence is the result of comparing C_R and C_A. In particular, given a particular set of dependencies among tasks, socio-technical congruence is the proportion of coordination activities that actually occurred (given by the *Actual Coordination* matrix) relative to the total number of coordination activities that should have taken place (given by the *Coordination Requirements* matrix). For example, if the *Coordination Requirements* matrix shows that 10 pairs should coordinate, and of these, 5 show *Actual Coordination* interactions, then the congruence is 0.5. Formally, we define congruence as follows:

$$\text{Diff}\,(C_R, C_A) = \text{cardinality}\,\{\text{diff}_{ij} \mid cr_{ij} > 0 \;\&\; ca_{ij} > 0\}$$

$$|C_R| = \text{cardinality}\,\{cr_{ij} > 0\}$$

we have,

$$\text{Congruence}\,(C_R, C_A) = \text{Diff}\,(C_R, C_A) / |C_R| \qquad (2)$$

The value of congruence belongs to the [0,1] interval that represents the proportion of coordination requirements that were satisfied through some type of coordination activity. The measure of socio-technical congruence proposed here provides a new way of thinking about coordination, particularly, by providing a fine-grain level of analysis of different types of product dependencies and allowing us to examine how coordination needs are impacted by them.

IV. RESEARCH QUESTIONS

Utilizing the socio-technical congruence framework, our study examines how the alignment between task dependencies and coordination activities impact traditional outcomes measures of software development projects. Specifically, we address the following general research questions:

RQ 1: What is the impact of congruence on development productivity and software failures?

RQ 2: How do coordination requirements evolve over time?

V. RESEARCH SETTING

We examined our research questions using data collected from two large scale software development projects from two distinct companies.

A. Project A: Distributed System

Project A was a large distributed system for data storage product. The data covered a period of 39 months of development activity and the first four releases of the product. The company had one hundred and fourteen developers grouped into eight development teams distributed across three development locations in North America. All the members of each team were collocated. All the developers worked full time on the project during the time period covered by the data. The system was composed of about 5 million lines of code distributed in 7737 source code files mostly in C language and a small portion (117 files and less than 96000 lines of code) in C++ language. The system consisted of 27 architectural components and the development responsibility of each component was assigned to one of the eight development teams. All developers had full access to a version control system and a modification request tracking system. The data cor-

responding to a total of 8,257 resolved modification requests (MRs) involving 67,652 commits to the version control system.

Software developers communicated and coordinated using various means. Opportunities for interaction existed when working in the same formal team or when working in the same location through periodic team meetings as well as impromptus interactions. Developers also used tools such as Internet Relay Chat (IRC) and a MR tracking system to interact and coordinate their work. Each formal team had a channel in IRC where most of the interactions with the team took place. The company stored the contents of those IRC channels and such content was viewable through a web-based interface. The MR tracking system kept track of the progress of the development task, comments and observations made by developers as well as additional material used in the development process such as snapshots of screen reporting errors, trace dumps or specification documents. We collected communication and coordination information from these two systems. Finally, we also collected demographic data about the developers such as the geographical location and the team he/she belonged to.

B. Project B: Embedded System

We collected data from a multi-national development organization responsible for producing a complex embedded system for the automotive industry. The development organization used a product line approach and the collected data covered 65 months of development activities associated with the latest version of the main platform software from the beginning of the project in late 2003 until end of 2008. Three hundred and eighty developers located in eight development sites distributed across Europe and India participated in the project. Those developers were organized in 37 development teams. The system was composed of about 7 million lines of code distributed in 9,074 source code files and 562 architectural components. The development responsi-

bilities of each component were assigned to a single development team. All source code files were written in C language. All developers have full access to the version control system and the modification request tracking system. The data corresponding to a total of 4,170 resolved modification requests were identified and 2,340 of those were multi-team modification requests. Those MRs involved 10,736 commits to the version control system.

The development organization had in place a well-defined development process. Telephone and conference calls as well as email were the primary mechanisms of communication among distributed engineers. The use of communication technologies such as instant messaging was not allowed. Finally, we also collected demographic data about the developers such as the geographical location and the team he/she belonged to.

VI. Software Dependencies and Coordination Requirements

The first step in our investigation examines how software dependencies can be utilized to extract coordination requirements among developers. Based on recent studies, we computed software dependencies by examining the set of source code files that are modified together as part of a modification request. This approach is equivalent to the approach proposed by Gall and colleagues (1998) in the software evolution literature to identify logical dependencies between modules. A source code file can be viewed as representing a "bundle" of technical decisions. If a modification request can be implemented by changing only one file, it provides no evidence of any dependencies among files. However, when a modification request requires changes to more than one file, it can be assumed that decisions about the change to one file in a modification request depend in some way on the decisions made about changes to the other files involved in implementing the modification request. Dependencies could range from syntactic, for instance a function call between files, to more complex semantic dependencies where the computations

done in one files affect the behavior of other files. This approach has been shown to represent a good estimate for a broad range of dependencies that affect productivity and defect-proneness.[1]

We now turn our attention to research question 1 which asks if we can determine the coordination needs among developers of a software project. The following paragraphs address this question in two parts. First, we provide an interpretation of the model in equation 1 that allows us to compute coordination requirements based on the socio-technical congruence framework described in section III. Second, we demonstrate how the coordination requirements can be calculated from data in traditional software repositories such as version control and change management systems.

A. From Software Dependencies to Coordination Requirements

The socio-technical congruence framework provides us with a measure of coordination requirements that consists of two elements: a people to task relationship (Task Assignment matrix) and task to task relationship (Task Dependency matrix). In the context of software development, we interpret the people to task relationship as developers modifying source code files. The second relationship can be articulated as the technical dependencies among files. Combining these two definitions with equation (1) allows us to compute coordination requirements. The outcome is a people to people matrix where each cell *ij* indicates the extent to which developer *i* is dependent of developer *j* given the set of logical dependencies of the system under development and the allocation of work responsibilities.

[1] For completeness, we computed all congruency measures using both logical dependencies and syntactic dependencies. Our results were in agreement with published literature (Cataldo et al, 2008; Cataldo, et al, 2009), with logical dependencies far more predictive of failures and producitvity. We omit these results in the interest of space.

B. Coordination Requirements from Software Repository Data

The data for constructing the *Coordination Requirements* matrix can be extracted from two types of repositories commonly encountered in software development projects: the modification request tracking systems and the version control systems. Modification reports in our data contained information about which commits in the version control system represent the development effort for that report. Thus, an MR provides the "developer *i* modified file *j*" relationship that constitutes our T_A matrix described in the previous section.

The *Task Dependency* (T_D) matrix was constructed as follows. The cell c_{ij} of the T_D matrix represents the number of times a particular pair of source code files changed together as part of the work associated with modification requests prior to the focal MR. Finally, using those T_A and T_D matrices, the *Coordination Requirement* (C_R) matrix can be computed as indicated earlier.

Figure 1 depicts an example of using the algorithm with data from software repositories. In this case, we have a modification request involving 3 developers and a system with 5 source code files. The T_A matrix captures the set of files that each developer modified as part of working on the modification request. For instance, developer 1 (top row) worked on 2 files (cells in the first row containing 1). The T_D matrix, in this case, contains logical dependency information. The diagonal indicates the total number time the source code files was changed in the past development tasks, while the off-diagonal cells indicate the number of times the two files were changed together as part of past modification requests. For example, the value 3 in $cell_{1,3}$ (top row, third column from left to right) indicates that files 1 and 3 were changed together three times in the past. Then, the resulting C_R matrix represents the extent to which developers are interdependent giving the history of how source code files were changed (T_D matrix) and the files that were modified as part of the development tasks considered in the T_A matrix. For instance, developers 1

and 3 are the highly coupled ($cell_{1,3}$ in C_R equals 10) whereas developers 1 and 2 are significantly less interdependent ($cell_{1,2}$ in C_R equals 1). Notice since the T_D matrix is symmetric the resulting C_R matrix is also symmetric.

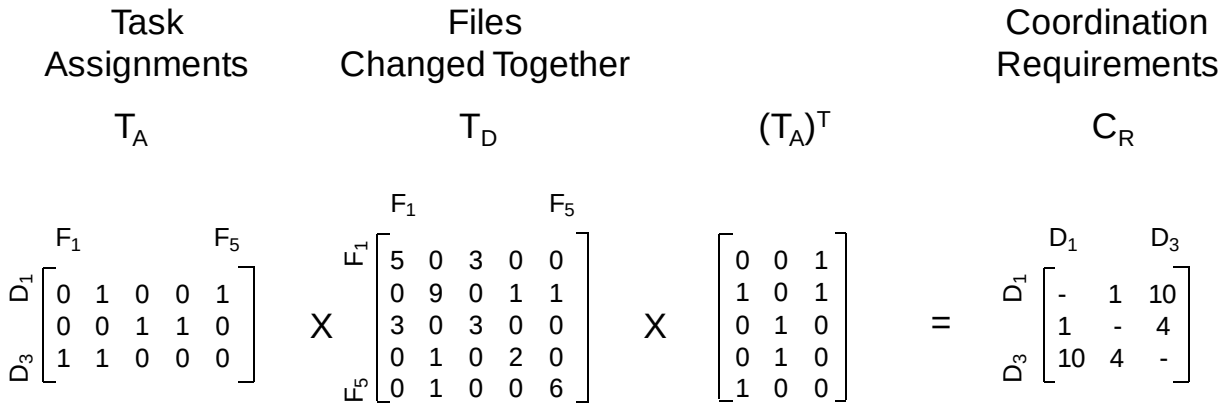

Figure 1: Example of the Computation of the Coordination Requirement Matrix

VII. Congruence and Software Development Productivity

The second step in our analysis consists of evaluating the relationship between socio-technical congruence and development time, an outcome that past research in software development has shown to be negatively impacted by failure to adequately coordinate development tasks (Curtis et al, 1988; Espinosa et al, 2007; Herbsleb & Mockus, 2003).

Empirical research has shown that difficulties in communication and coordination breakdowns are particularly problematic when the work items involve more than one team (Curtis et al, 1988; Espinosa et al, 2007; Kraut & Streeter, 1995) and the developers are geographically distributed (Herbsleb & Mockus, 2003). For these reasons, the analysis focuses on the set of modification requests that involved more than one software development team. A total of 2,375 multi-team modification requests were identified in project A. Project B contained 2,480 multi-team modification requests.

A. Description of the Measures

Past research has identified a number of factors that impact development time. Some of those factors are related to characteristics of the development tasks such as the amount of code to be written or modified and the priority of the task, whereas other factors capture relevant attributes of the individual developers and the teams that participate in a development task. In the following paragraphs, we first describe our dependent variable, resolution time of modification requests. Secondly, the procedures used to construct the measures of congruence are described. Finally, we describe a number of additional control measures that were included in the statistical models.

1) Development Time Measure

Our measure of development time is *Resolution Time,* which captures the time it took to resolve a particular modification request, accounting for all the time that the MR was assigned to developers. The modification requests reports contain records of when the MR was opened and resolved as well as every time the MR was assigned to a particular developer. Given this information, we can approximate the amount of time that developers were actually working on the task.

2) Congruence Measures

We collected logical dependency information from both projects in order to compute the coordination requirement matrices. Logical dependencies were extracted from the modification request tracking and version control systems in both projects as described in section VI-B.

Development organizations have different ways of coordinating their activities, that is, they have different sets of coordination capabilities. In the measure of congruence described in section III, the *Actual Coordination* (C_A) matrix captures one or more coordination capabilities of a

particular organization. We constructed two distinct C_A matrices for each project, which led to two distinct measures of congruence. One C_A matrix that we call *Structural Congruence,* captured the existence of coordination activity between engineers *i* and *j* if they belong to the same formal team. Such a matrix represented the potential paths of communication and coordination that members of a formal team have through various mechanisms such as team meetings and other work-related activities. This particular view of congruence captures the essence of the theorized relationship between product and work modularization. If the system is modular then the work in one module can be done within one team, then no coordination is required among different teams.

Similarly to the case of organizational structure, the second C_A matrix that we call *Geographical congruence* was built around the idea of potential paths of communication and coordination that exist when individuals work in the same physical location (Allen, 1997; Olson & Olson, 2000). Then, in terms of the matrix of coordination activities, engineers *i* and *j* have a linkage if they work in the same location.

In the case of project A, we had data concerning two additional communication and coordination mechanisms: the exchanges of information in the MR tracking system and IRC. *MR communication congruence* considers an exchange of technical information between engineers *i* and *j* only when both *i* and *j* explicitly commented in the modification request report. Multiple modification requests might refer to the same problem and later be marked as duplicates of a particular modification request. All duplicates of the focal MR were also used to capture the interactions among developers. Finally, *IRC communication congruence* was computed based on interaction between developers from the IRC logs. Three raters, blind to the research questions, examined the IRC logs corresponding to the period of time associated with each MR and established an

interaction between engineers *i* and *j* if they made reference to the MR identifier or to the task or problem represented by the MR in their conversations. In order to assess the reliability of the raters' work, 10% of the MRs where coded by all raters. Comparisons of the obtained networks showed that 98.2% of the networks had the same set of nodes and edges.

3) Additional Control Measures

Past research has reported several additional factors that impact development time (Espinosa et al, 2007; Herbsleb & Mockus, 2003; Kraut & Streeter, 1995). We collected a number of control variables that capture attributes of the development tasks, the individuals and the teams associated with the development work. The amount of code written or changed is a proxy for the actual amount of development work done. The *change size* measure was computed as the number of files that were modified as part of the change for the focal MR. Prior research (Espinosa et al, 2007) has used lines of code changed as a measure of the size of the modification; however, a comparative analysis of both measures showed equivalent results in the statistical model used in this study. Therefore, the results presented in this study are based on the measure computed from the number of files modified. The change size measure was highly skewed so a log transformation was applied to satisfy the normality requirements of the regression model used in our analysis (see table A1 in the appendix).

Two measures were constructed to capture attributes of the teams involved in each modification request. *Team load* is a measure of the average work load of the teams responsible for the components associated with the modification request. This control variable was computed as the ratio of the average number of modification requests in open or assigned state over the total number of engineers in the groups involved in the focal modification request during the period of time the MR was in assigned state. *Multiple Locations* is a binary variable that indicates whether

all the developers that worked on a particular MR were in the same geographical location (a value of 0) or were distributed across different development locations (a value of 1).

An experienced software engineer familiar with tools and programming languages can be substantially more productive than an inexperienced developer (Brooks, 1995; Curtis, 1981; Curtis et al, 1988). Furthermore, experience with the domain area and the technical characteristics of the application being developed help accelerate development time (Curtis et al, 1988). We used archival information as well as data from the software repositories to compute several individual level measures of experience. First, *Component experience* was computed as the average number of times that the engineers responsible for the modification request have worked on the same files affected by the focal modification request. Second, *Shared Work Experience* was computed as the average number of times both persons in each dyad in a modification request worked together prior to the focal MR, averaged across all dyads. Both experience measures were log-transformed to satisfy normality requirements (see table A1 in the appendix).

B. Description of the Statistical Model

Past research has found that linear (Espinosa et al, 2007; Herbsleb et al, 2006) and hierarchical linear (Espinosa et al, 2007; Kraut & Streeter, 1995) models are appropriate techniques for examining the effects of different factors on development productivity. In this study, we examined the effect of the various congruence measures on task performance using the following linear regression model:

$$ResolutionTime = \sum_i \beta_i * CongruenceMeasure_i + \sum_j \delta_j * ControlVariable_j + \varepsilon \quad (3)$$

We report several measures of fit for each linear regression model including R^2, the adjusted R^2 and the F-statistic as indicators of the models' explanatory power. An examination of descrip-

tive statistics and Q-Q plot indicated that several of the variables (*Resolution Time*, *Chang Size*, *Shared Work Experience* and *Component Experience*) were highly skewed to the left. The log transformation provided the best approximation to a normal distribution (see table A.1 in the appendix). An analysis of the pair-wise correlations amongst the independent variables as well as a variance inflation analysis suggested no relevant collinearity problems. A small set of correlations were statistically significant but their levels did not exceed +/- 0.33 (see table A.3 in the appendix).

C. Results

We performed several linear regression analyses to assess the effect of the congruence measures on resolution time. Table I reports the OLS coefficients from the various regression models. We constructed a baseline model for both projects (model I and IV) considering only the control factors. The results are consistent with previous empirical work in software engineering. Factors such as the size of the modification, familiarity with the software components, familiarity working together with other developers and are geographic distribution are significant factors impacting the resolution time of modification requests (Boh et al, 2007; Espinosa et al, 2007; Herbsleb & Mockus, 2003).

We then included the measures of structural and geographical congruence based on logical dependencies. The results are reported in models II and V for project A and B, respectively. Both measures of congruence, structural and geographical, have statistically significant effects and the negative values of the estimated coefficients suggest that higher levels of congruence are associated with a reduction in time to resolve a modification request. More specifically, the association of higher levels of structural congruence with shorter development times supports the argument that when coordination requirements are contained within a formal team, task performance

increases. Geographical congruence had a positive effect on resolution time, consistent with past research finding distance has detrimental effects on communication (see Herbsleb & Mockus, 2003 and Olson & Olson, 2000 for reviews).

The measures of structural and geographical congruence could be affected by personnel turnover and mobility across teams. In project A, we examined archival data collected from the company and we determined a yearly turnover rate of only 3% and an inter-group mobility rate of less than 1%. The modification requests that involved individuals that left the company or changed group membership were eliminated from the analysis. However, an analysis including those modification requests showed results consistent with those reported in tables 1 and 3. Unfortunately, we did not have access to the necessary data in project B.

TABLE I
The Impact of Congruence on Resolution Time of Modification Requests

	Project A		**Project B**	
	Model I	**Model II**	**Model III**	**Model IV**
(Intercept)	3.681**	4.301**	4.218**	4.799**
Change Size (log)	0.402**	0.308**	0.331**	0.213**
Team Load	0.014	0.013	-0.011	-0.004
Multiple Locations	0.523**	0.527**	0.282**	0.234*
Shared Work Experience (log)	-0.144**	-0.146**	-0.116**	-0.117**
Component Experience (log)	-0.141**	-0.141**	-0.213**	-0.254**
Structural Congruence		-0.239*		-0.329*
Geographical Congruence		-0.382*		-0.360*
Model Fit				
N	2375	2375	2480	2480
R^2	0.652	0.692	0.459	0.522
Adjusted R^2	0.651	0.691	0.457	0.521
F-test	887.7**	787.1**	419.8**	385.6**

($^{+}$ p < 0.10, * p < 0.05, ** p < 0.01)

The data collected in project A allowed us to compute two additional measures of congruence, IRC and MR communication congruence. Table II reports the results from the models that included all four measures of congruence. The results are consistent with those reported in table I. In addition, congruence based on the coordination activities amongst engineers performed through the MR reports as well as IRC were also statistically significant suggesting the useful-

ness of these tools as additional coordination capabilities that facilitate interdependent work, particularly in a geographically distributed context.

TABLE II
THE IMPACT OF CONGRUENCE ON RESOLUTION TIME WHEN CONSIDERING ADDITIONAL COORDINATION CAPABILITIES IN PROJECT A

	Model I	**Model II**
(Intercept)	3.681**	4.278**
Change Size (log)	0.402**	0.307**
Team Load	0.014	0.013
Multiple Locations	0.523**	0.528**
Shared Work Experience (log)	-0.144**	-0.145**
Component Experience (log)	-0.141**	-0.141**
Structural Congruence		-0.235*
Geographical Congruence		-0.372*
MR Congruence		-0.146*
IRC Congruence		-0.154*
Model Fit		
N	2375	2375
R^2	0.652	0.744
Adjusted R^2	0.651	0.743
F-test	887.7**	763.7**

($^{+}$ p < 0.10, * p < 0.05, ** p < 0.01)

VIII. CONGRUENCE AND SOFTWARE FAILURES

Many software failures are caused by dependencies that are not recognized by the engineers that architect, design and implement a software system (de Souza et al, 2004; Cataldo et al, 2009). This section examines the relationship between socio-technical congruence and failure proneness defined as the likelihood of software entities (e.g. source code files, modules or components) of being modified as part of fixing a field defect.

A. Description of the Measures

The literature has identified a number of factors that impact failure proneness of source code files. Some of those factors are related to attributes of the files, attributes of the technical dependencies among the files as well as characteristics of the development activities that modified those source code files. In the following paragraphs, we first describe our dependent variable, *File Buggyness*. Secondly, the procedures used to construct the measures of congruence are de-

scribed. Finally, we describe a number of additional control measures that were included in the statistical models.

1) Measuring Failure

The dependent variable, *File Buggyness*, is a binary measure indicating whether a file has been modified as part of the resolving a field defect. Therefore, the unit of analysis is the source code file. In project A, our data covered four releases of the product. Defects reported after the official release of the product were considered field defects. In the case of project B, the consequences of field defects are quite severe because they typically involve recall of vehicles with significant financial and reputational impact. A great effort goes into quality assurance and, consequently, field defects seldom occur. Therefore, we considered "field defects" as those defects encountered during the integration and system testing phase. Using 2,375 modification requests from project A and 4,170 modification requests from project B, the dataset of source code files was constructed in the following way. First, the dataset included all the files that were modified as part of the development activities of each project. For each one of those files, we determined if they were associated with a field defect in any of the releases of the product covered by the data. Secondly, we included all files that were associated with field defects that did not change during the development of a release under study. This procedure selection bias concerns.

2) Congruence measures

In order to construct the congruence measures, we used the same procedure described in section VI. However, the outcome in is a measure of congruence per MR. The congruence measures at the file level were constructed in the following way. For each file, we identified all the modification requests that touched the file. Then, the median value of each congruence measure was associated with that particular file. An alternative approach would be to compute the measure as

an average of level of congruence across all modification requests that affected each particular file. However, the number of MRs that affected a file was highly correlated with other measures and it was also a significant predictor of failures. Hence, it was more appropriate to use the measure based on the median value of congruence. As we indicated earlier, coordination data over the MR tracking systems and IRC were only available for project A, we computed only structural and geographical in the case of project B.

3) Additional Control Factors

Past research on failure proneness has identified a number of technical and work-related factors that impact it. On the technical dimension, Graves and colleagues (2000) suggested that such measures could be grouped in two sets: process measures and product measures. Process measures such as number of previous faults, number of deltas, age of the code and the number of developers that modified the files have been shown to be very good predictors of failures (Graves et al, 2000; Nagappan & Ball, 2007). These measures are also referred to in the literature as churn metrics. In terms of product measures, investigations have typically focused on measures such as size of the code and complexity measures and the empirical results are mixed. Researchers have found a positive relationship between lines of code and failures (Briand et al, 2000; Graves et al, 2000). However, other work has found a negative relationship between lines of code and failures, that is, a larger number of LOC decreases the likelihood of failures (Basili & Perricone, 1984). More recently, researchers have also looked at the role of software dependencies on failure proneness (Cataldo et al, 2009; Nagappan & Ball, 2007; Nagappan et al, 2008; Zimmerman & Nagappan, 2008).

The work by Cataldo and colleagues (2009) examined the relationship of failure proneness with several process, product and work-related metrics in two large scale industrial software de-

velopment projects. The authors performed several collinearity diagnostics and identified a subset of technical and work related measures to be included in the statistical models. Those measures constitute the set of control variables used in this study and they are described in the subsequent paragraphs. The size of the file (*LOC*) as the number of non-blank non-comment lines of code. We also computed for the *Average Number of Lines Changed* in a file as part of modification requests. Using the information in the T_D matrices discussed in section VI, we computed the two measures proposed by Cataldo and colleagues (2009). *Number of logical Dependencies* was computed from the corresponding T_D matrix as the row sum minus the cell in the diagonal. Notice that the T_D matrix based on logical dependencies is symmetric, therefore, the variable could also be computed using a summing the cells in the corresponding column. *Clustering of Logical Dependencies* measure for file *i* was computed as the density of connections among the direct neighbors of file *i*. This measure is equivalent to Watts's (1999) local clustering measure. Finally, we used equation 1 to construct the *Coordination Requirement* measure. A modification request provides the "developer *i* modified file *j*" relationship. We grouped such information across all modification requests in a particular release of the product to construct the T_A matrix. The T_D matrix was the same used for the computation of the logical dependency measures. Then, the *Coordination Requirements* measure captures for each file *i*, the degree centrality of the most central developer in the C_R matrix that worked on the file *i*.

B. Description of the Statistical Model

Since our dependent measure is a binary variable, our analyses used the logistic regression model described in equation 4 to assess the impact of socio-technical congruence on failure proneness.

$$P(Buggyness = 1 \mid X) = [1 + e^{(\sum_i \beta_i * CongruenceMeasure_i + \sum_k \varphi_k * ControlMeasure_k)}]^{-1} \quad (4)$$

where X is the vector consisting of the congruence and control measures. As it is customary with logistic regressions, the models were estimated using a maximum-likelihood method. We report several goodness-of-fit measures for each statistical model χ^2, the percentage of deviance explained by the model as well as the statistical significance of the difference between a model that adds new factors and the previous model without the new measures. Deviance is defined as -2 times the log-likelihood of the model. The percentage of the deviance explained is a ratio of the deviance of the null model (containing only the intercept), and the deviance of the final model.

Traditionally, regression coefficients are reported as part of the results. However, since we are using logistic regressions, we decided to report odds ratios because they simplify the interpretation of the results. Odds ratios are the exponent of the logistic regression coefficient. Odds ratios larger than 1 indicate a positive relationship between the independent and dependent variables whereas an odds ratio less than 1 indicates a negative relationship. For instance, an odds ratio of 2 associated with the number of logical dependency measure indicates that a unit change in such measure doubles the probability of a file having a customer reported defect when the remaining factors in the model are kept constant.

C. Results

We performed several logistic regression analyses to assess the effect of the congruence measures on failure proneness of source code files. Table III reports the odds ratios associated with the various regression models. We constructed a baseline for both project A and B (models I and III). The results from both projects are consistent with those reported by Cataldo and colleagues (2009). Models II and IV introduce the measures of congruence based on logical dependencies for projects A and B, respectively. The results show a significant impact of both structural and geographical congruence in project B (model IV) with higher levels of congruence being asso-

ciated with lower levels of failure proneness (odds ratios less than 1). In project A, only structural congruence was statistically significant.

The data collected in project A allowed us to compute two additional measures of congruence, IRC and MR communication congruence. Table IV reports the odds ratios from the models that included all four measures of congruence. The results from model II are consistent with those reported in table III. In addition, IRC and MR congruence are also statistically significant suggesting such means of communication and coordination are very valuable in managing dependencies and, consequently, improving software quality. Model III in table II also shows no statistically significant effect for the measures of congruence computed based on syntactic dependencies.

It is worth mentioning that we also replicated the analyses using the data from the entire set of MR from project B (8,257 MRs) and the results were consistent to those reported in table III. However, we did not have IRC-based coordination data for the entire set of modification requests. Consequently, we decided to report the analyses based on a dataset that allows us to examine the impact of all four measures of congruence.

TABLE III
THE IMPACT OF CONGRUENCE ON FAILURE PRONENESS

	Project A		**Project B**	
	Model I	**Model II**	**Model III**	**Model IV**
LOC (log)	1.125**	1.137**	1.251**	1.151**
Avg. Lines Changed (log)	1.128**	1.118**	1.331**	1.314**
Number Logical Dep. (log)	2.219**	2.109**	3.829**	3.830**
Clustering Logical Dep. (log)	0.012**	0.012**	0.002**	0.002**
Coordination Req. Dep. (log)	2.187**	1.967**	1.111**	1.801**
Structural Congruence		0.285*		0.859*
Geographical Congruence		0.317		0.578*
Model Fit				
N	3980	3980	9074	9074
Model $\chi 2$	1663**	1701**	3092**	3401**
df	5	7	5	7
Deviance Explained	0.302	0.317	0.362	0.401
Model Comparison $\chi 2$	--	38.13**	--	309.07**

(+ $p < 0.10$; * $p < 0.05$; ** $p < 0.01$)

Summarizing, the results of our analyses suggest that the relevant work dependencies that impact software failures stem from logical dependencies rather than syntactic relationships between software entities. Furthermore, the various measures of congruence demonstrate the complementary benefits of different approaches of satisfying coordination needs which if left unattended can results in poorer software quality.

TABLE IV
THE IMPACT OF CONGRUENCE ON FAILURE PRONENESS WHEN CONSIDERING ADDITIONAL COORDINATION CAPABILITIES IN PROJECT A

	Model I	**Model II**
LOC (log)	1.125**	1.136**
Avg. Lines Changed (log)	1.128**	1.121**
Number Logical Dep. (log)	2.219**	2.109**
Clustering Logical Dep. (log)	0.012**	0.012**
Coordination Req. Dep. (log)	2.187**	1.962**
Structural Congruence (Logical Dep.)		0.281*
Geographical Congruence (Logical Dep.)		0.317
MR Congruence (Logical Dep.)		0.471**
IRC Congruence (Logical Dep.)		0.044**
Model Fit		
N	3980	3980
Model $\chi 2$	1663**	1859**
df	5	9
Deviance Explained	0.302	0.335
Model Comparison $\chi 2$	--	196.24**

(+ $p < 0.10$; * $p < 0.05$; ** $p < 0.01$)

IX. THE TEMPORAL EVOLUTION OF CONGRUENCE

The various measures of congruence based on logical dependencies are associated with improvements in development productivity and software quality. However, those analyses do not examine the role of two important factors that could impact a development organization's ability to effectively coordinate: learning and product evolution. It is well established that groups and organizations are able to learn over time to perform their task better (Argote, 1999). In our context, as developers gain experience in different aspects of the development project such as how the various components of the system being developed relate to each and who is working in the various parts of a system, we could expect that developers, over time, are better able to identify

and manage emerging work dependencies. However, the benefits accrued from learning can be diminished when important changes in the coordination requirements take place (e.g. structure of the system or the allocation of functional responsibilities change). The socio-technical congruence framework allows us explore these issues by examining how the congruence measures evolve over time. The rest of this section examines the evolution of congruence in both projects, A and B.

In order to examine the evolutionary patterns of congruence, we constructed measures of congruence on a semi-annual basis which resulted in 6 semesters for project A and 10 semesters for project B. We considered all the modification requests started and completed within each semester of each projects covered by our data. Combining that data with the logical dependencies data corresponding to the same semester, we constructed a coordination requirements (C_R) matrix for each semester. Then, the congruence measures for each semester were calculated using the same approach discussed in section VI. Figure 2 shows the average level of the structural and geographical congruence over time in both projects. We observe that in project A (continuous lines), structural congruence declines significantly over time. A similar pattern also occurs in project B (dashed lines) but the magnitude of the decline is not as severe as in project A. The decline in structural congruence could be interpreted as a deterioration of the homomorphic relationship between product and work structures posited by the modularity theoretical argument (Baldwin & Clark, 2000; Conway, 1968; Parnas, 1972). On the other hand, we observe that the geographical congruence remains relatively stable in both projects suggesting that the coordination needs across locations did not change over time. However, the low levels of congruence, particularly in project B, suggest that the organization was never able to appropriately deal with cross-site dependencies.

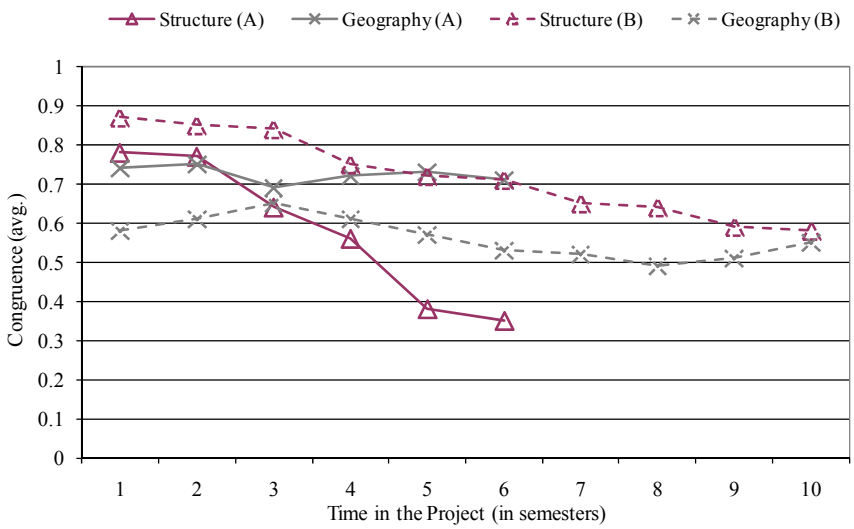

Figure 2: The Evolution of Structural and Geographical Congruence

In the case of project A, we can compare the evolution of all four measures of congruence. Figure 3 depicts such patterns and we observe that structural and geographical congruence dominate in the earlier semesters while communication congruence based on MRs or IRC are almost absent. In later semesters, structural congruence decreases significantly, particularly after the 3rd semester, while the measures of communication congruence based on MR and IRC increase significantly, remaining high during the last two semesters. These patterns suggest a learning effect where developers substitute the lack of formal communication and coordination paths with coordination through other means of communication such as IRC and MR reports.

The changes in the various measures of congruence depicted in figures 2 and 3 raise a follow up question: are all developers able to identify the changes in coordination requirements and adapt their coordinative actions accordingly? Mockus and colleagues (2002) and Cataldo and colleagues (2006) reported that in open source and commercial projects a large portion of the modifications to a software are made by a small number of developers. Following such insight, we examined the patterns of contributions in both projects. In project A, we found that 50% of the modifications made to the software system were done by only 18 (15%) developers. In

project B, 107 developers (28%) contributed 50% of the modifications. Based on these results, we separated the developers in two groups: top performers and the rest[2].

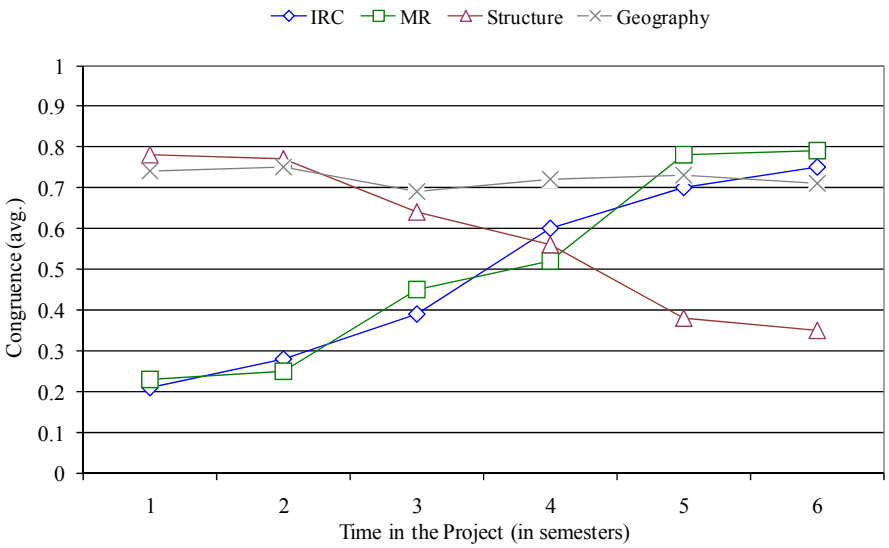

Figure 3: The Evolution of all Measures of Congruence in Project A

Figure 4 shows the evolution of the congruence measures for both groups in project A. The top performers are represented on left hand side of the graph (solid lines) while the rest of the developers are represented on the right hand side of the graph (dashed lines). The patterns are surprisingly different. Top-performers were involved in tasks that required significantly more coordination across teams (decreasing structural congruence) and, over time, they became quite effective at coordinating over IRC and the MR tracking system. On the other hand, the rest of the developers seem not to use the computer-mediated communication tools (e.g. IRC) to interact with the right set of people. Consequently, they never achieve high levels of congruence in the IRC and MR congruence measures. Figure 5 depicts the evolution of structural and geographical congruence for both groups of developers, top-performers and the rest, in project B. We observer a

[2] We performed several statistical comparisons and the two groups of individuals in both projects did not differ in terms of experience, familiarity working together, the average size of the changes made to the software and the average number of lines added or removed from the software.

similar pattern as in project A where top-performers tend to be involved in tasks that require cross-team coordination, particularly as the project matures.

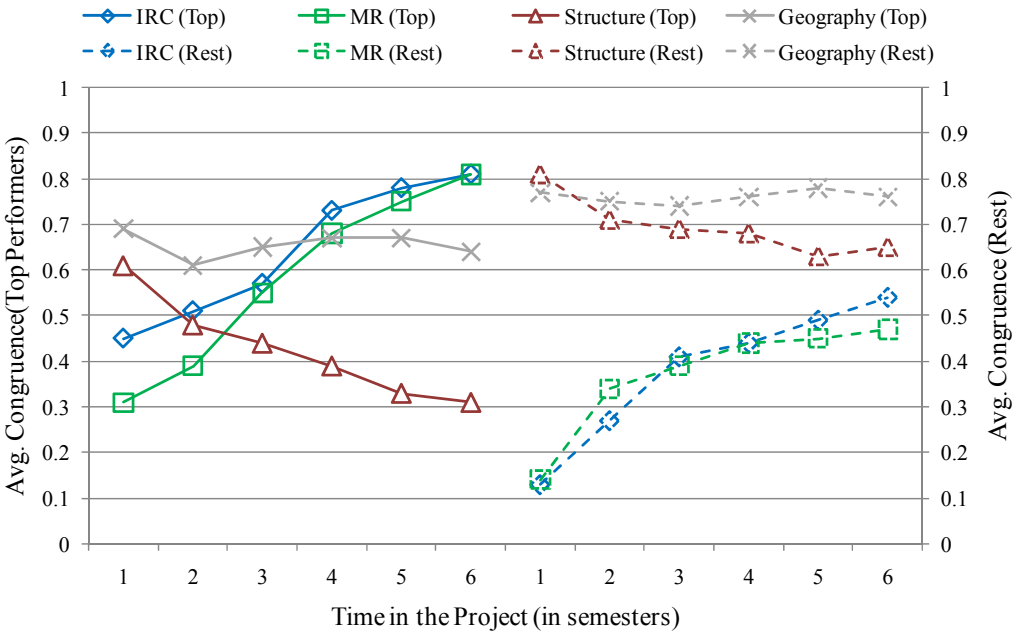

Figure 4: The Evolution of Congruence for Top-Performers and the Rest of the Developers in Project A

Summarizing, the section presents two additional and important results. First, the decreasing levels of structural congruence suggest that the benefits of modularization diminish as a project matures. Second, high performing developers tend to exhibit very different coordinative actions relative to less productive individuals suggesting that the traditional perspective in software engineering relating only cognitive ability and experience to contributions (Curtis, 1981) may not capture all of the important attributes of top-performing developers, since their performance seems to have a substantial social component.

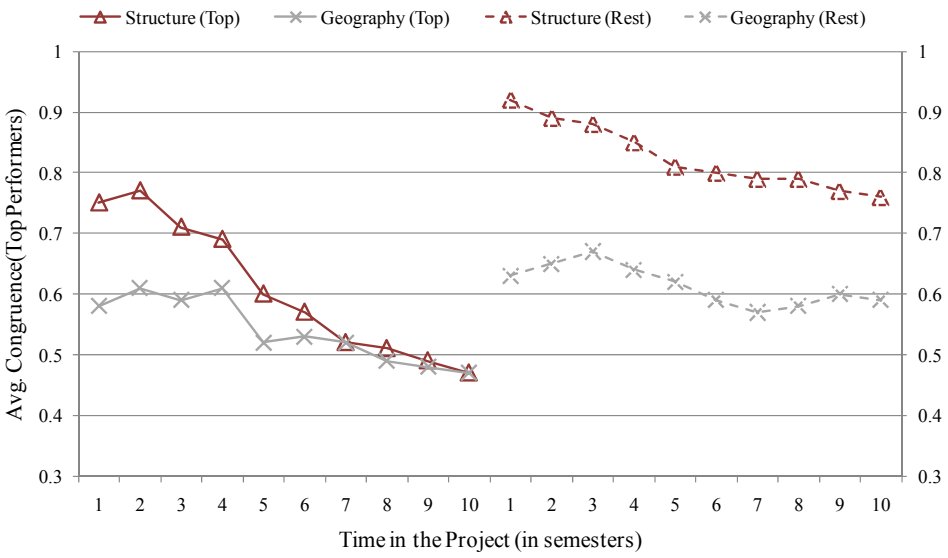

Figure 5: The Evolution of Congruence for Top-Performers and the Rest of the Developers in Project B

X. COORDINATION REQUIREMENTS AND THEIR TEMPORAL EVOLUTION

The step in our investigation consisted on examining how the coordination needs of the development organization evolve over time. This is a particularly important factor because highly volatile coordination needs render traditional coordination approaches inadequate. We focused our attention on two key aspects of the coordination requirements: the average change in an individual's coordination needs and the amount of coordination needed that crosses team boundaries. Figure 6 depicts the evolution of the average change in an individual's coordination needs in projects A and B on a monthly basis over the time period covered by the datasets (39 months for project A and 65 months for project B). The average change in an individual's coordination needs is computed by comparing the coordination requirements matrices (constructed using the logical dependencies) from month *t* against month *t-1* and averaging the amount of change in the coordination needs across all the individuals. As an example, a 10% value of change in the coordination needs in month *t* means that 10% of the coordination requirements of any particular de-

veloper did not exist in the previous month (*t-1*). The amount of coordination requirements that involve developers from other formal organizational groups is depicted in figure 6 for both closed source projects. The values are computed by identifying those coordination requirements that cross the boundaries of an individual's team and averaging those specific coordination needs. Figure 6 shows significant volatility in the coordination requirements over time. There are several instances where the level of change in the coordination requirements is above 20% for both projects. Specifically, the average change in the developers' coordination needs across all months in project A is 12.9% (minimum = 2.2% and maximum = 44.1%) while project B exhibit an average of 13.5% across all 65 months covered by the data (minimum = 3.5% and maximum = 27.9%). Considering those percentages in terms of actual number of distinct individuals that a developer needs to coordinate with, on average and on a monthly basis, we have 6.8 different developers in project A and 21.5 different developers in project B. A similar pattern also occurs for the amount of coordination needs that cross the team boundary as depicted in Figure 7. In this case, we see an average change in coordination requirements across team boundaries of 15.2% in project A (minimum = 4.5% and maximum = 35.7%) and 11.6% (minimum = 0.2% and maximum = 22.4%) for project B.

In sum, the results presented in this section show that coordination requirements tend to be quite volatile over the life of a software development project. More importantly, coordination needs that cross the formal organizational boundaries (e.g. formal team) are also quite common. Combined, these results suggest the need to new coordination mechanisms such as collaborative tools that are able to assist developers navigate the complex landscape of dependencies that impact their work.

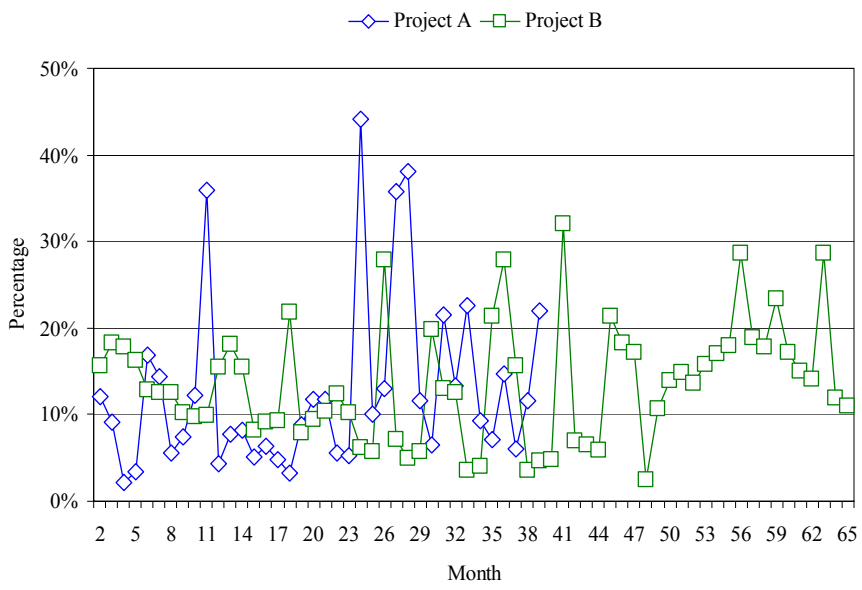

Figure 6: The Average Change in Coordination Requirements on a Monthly Basis

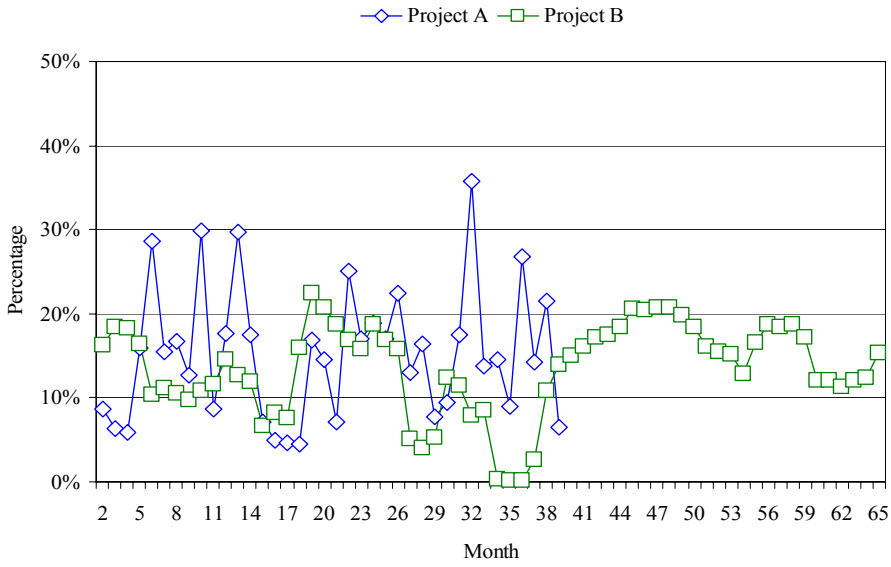

Figure 7: The Average Change on Out-Group Coordination Needs on a Monthly Basis

XI. DISCUSSION

This paper introduced the notion of socio-technical congruence as a framework for examining fine-grain technical dependencies among software entities (e.g. source code files, modules, com-

ponents) and their implications for coordination needs in development organizations. We addressed three specific research questions. First, we proposed a technique to compute coordination requirements among developers based on the technical dependencies among the parts of a software system modified in the development tasks (research question 1). We evaluated the proposed approach to calculate task dependencies in two large scale software development projects. The results of our empirical analyses revealed that the gaps between the computed coordination requirements and the actual coordination activities carried out by the developers had major implications on development productivity and software quality (research question 2). In particular, our analyses showed that when engineers identified and managed the *relevant* coordination needs, development productivity and software quality improved.

Finally, research question 3 was addressed by examining the evolution of the coordination requirements as well as the evolution of the mismatches between coordination needs and coordination behavior (congruence) over the lifetime of the projects. Our analyses revealed two important results. First, our measure of structural congruence, which captures how well the organizational structure of a software project supports the coordination needs stemming from the technical structure of the system, diminishes over time. Such a pattern suggests a more complex relationship between the product and the organizational structures than theorized in the modular design literature (e.g. Baldwin & Clark, 2000). As demonstrated in our results, high levels of structural congruence (indicative of a good match between product and organizational structure) are associated with better productivity and quality complementing the abundance of evidence highlighting the benefits of modular designs. However, as the development of a system evolves, new dependencies emerge and they tend not to be aligned with the existing formal organizational structure. Thus, additional coordination capabilities are required. For instance, our analyses revealed

developers in project A tended to coordinate more congruently over communication mediums such as IRC and the MR tracking system as the coordination mechanism provided by the formal organization increasingly failed to match the coordination needs of the developers.

A second important result related to research question 3 relates to the evolution of the coordination requirements. Our qualitative analysis showed that coordination needs tend to be quite volatile. Moreover, we found that coordination requirements that cross organizational boundaries (e.g. formal teams) are also quite common. Combined, these findings demonstrate the dynamic nature of work dependencies that exist in software development. Individuals tend have difficulties identifying task interdependencies that are not obvious or explicit (de Souza et al, 2004; Sosa, 2008; Sosa et al, 2004) and the developers' ability to recognize dependencies diminish as coordination requirements change over time because communication and coordination patterns become embedded in the organizational practices (Henderson & Clark, 1990). In addition, coordinating across organizational boundaries is quite challenging and the negative impact of failed coordination on productivity is quite significant (Hoegl et al, 2004). For these reasons, volatility in the coordination requirements represents a major hurdle for software development projects, particularly, for those that are geographically distributed.

Future Research Directions

The results reported in this paper provide a number of directions for future research activities. First, our results suggest that collaboration tools could constitute an important coordination capability to support the development organization manage the dynamic and evolving web of work dependencies in software development projects. A host of collaboration and awareness tools have been proposed in the software engineering and CSCW literatures (e.g. Ripley et al, 2007; Schummer & Haake, 2001; Sarma et al, 2003; Sarma et al, 2009; Trainer et al, 2005). The meas-

ures of software dependencies used in our empirical analyses could provide complementary and, potentially more valuable, dependency information to such tools[3]. Particularly, in terms of raising awareness among developers' activities and the potential impact of their changes in the software system and other developers' work. In a more general context, traditional collaboration tools, such as email and instant messaging, which have become an integral part of work in the vast majority of organizations (Bellotti et al, 2003; Wattenberg et al, 2005) could also be improved. The coordination requirements measure could provide a way of identifying the email exchanges that are more relevant given the task interdependencies among individuals. This information would enable tools to present an enhanced task management experience by, for instance, prioritizing to-do-lists and generating reminders to respond to task-specific emails based on the coordination requirements. This email sorting approach could be thought as a task-specific alternative to other social-based sorting techniques such as the one proposed by Fisher and colleagues (Fisher et al, 2006). A more recent set of tools, such as sidebars (Cadiz et al, 2002) and productivity assistants (Geyer et al, 2007), would also benefit from the congruence framework. These types of tools focus on activity-centric collaboration and, as argued by Geyer and colleagues (2007), the majority of the tools assume user intervention in terms of deciding what type of information to make part of the sidebar. The congruence framework would provide an automatic mechanism to identify people of interest giving a particular set of task dependencies among the workers. Finally and in the context of large-scale software development projects such as the ones studied in the paper, the coordination requirement measure provides a mechanism to augment awareness tools that provide real-time information regarding the likely set of workers that a par-

[3] In order to take advantage of the congruence measures, tools would have to be able to identify (1) the set of source code files (or other software entities of interest) that were changed as part of a modification request and (2) the developers that made those changes.

ticular individual might need to communicate with. For instance, integrated development environments, such as Eclipse (2009) or Jazz (2009), could use the coordination requirement information to recommend a dynamic "coordination buddy list" every time particular parts of the software are modified. In this way, the developer becomes aware of the set of engineers that modified parts of the system that are interdependent with the one the developer is working on. The concept of the "buddy list" in communication and collaboration tools is not a new idea. However, the novel contribution is to construct the "buddy list" from accurate estimates of the set of individuals more likely to be relevant to a particular developer in relations to the work dependencies, information which is captured by the coordination requirements measure.

Second, the impact of congruence on productivity and quality highlight the importance of identifying potential coordination needs as early as possible in the development process in order to provide the development organization with the appropriate communication and coordination mechanisms. Unfortunately, identifying logical dependencies as proposed by our measures is not possible in early stages of a project where no source code has been developed. We see at least two research paths which deserve future attention. One path is to consider a *coordination* view of the architectures that combines the product's technical dependencies with relationships among the organizational units responsible for carrying out the development work. In order to generate such representations, methods of identifying relevant dependencies from the technically focused views of the architecture are to be devised. One potentially promising approach is to synthesize the dependencies represented in the various types of UML diagrams (e.g. class diagrams, sequence diagrams, collaboration diagrams, etc) into a single set of technical relationships among modules. Such a method might be able to identify logical relationships among parts of the systems which, as shown in this paper, are an important factor driving the work dependencies in

software development organizations.

Third, the proposed framework opens the door for the analyses of numerous other forms of technical dependencies such as grouping dependencies around product features or around "cross-cutting" concerns (Kiczales & Menzini, 2005), which might not necessarily match the source code file, module or component based representations of technical dependencies, and examine their implications on coordination. On the organizational side, we could extend the analyses to temporary organizational entities such as task forces, performance or integration teams) and examine their implications for coordination.

ACKNOWLEDGMENT

The authors also gratefully thank A. Hassan and R. Holt for providing the source code for their C-REX tool.

REFERENCES

Allen, T.J. (1977). *Managing the Flow of Technology*. MIT Press.

Baldwin, C.Y. and Clark, K.B. (2000). *Design Rules: The Power of Modularity*. MIT Press.

Bellotti, V., Ducheneaut, N., Howard, M., Smith, I. (2003). Taking email to task: the design and evaluation of a task management centered email tool. In *Proceedings International Conference on Human Factors in Computing Systems (CHI'03)*, Ft. Lauderdale, FL.

Basili, V.R. and Perricone, B.T. (1984). Software Errors and Complexity: An Empirical Investigation. *Communications of the ACM*, Vol. 12, No. 1, pp. 42-52.

Bass, M., Bass, L., Herbsleb, J.D. and Cataldo, M (2006). Architectural Misalignment: an Experience Report. To appear in the *Proceedings of the 6th International Conference on Software Architectures (WICSA '07)*.

Boh, W.F., et al. (2007). Learning from Experience in Software Development: A Multilevel Analysis. *Management Science*, Vol. 53, No. 8, pp. 1315-1331.

Briand, L.C., Wust, J., Daly, J.W. and Porter, D.V. (2000). Exploring the Relationships between Design Measures and Software Quality in Object-Oriented Systems. *The Journal of Systems and Software*, Vol. 51, pp. 245-273.

Brooks, F. (1995). *The Mythical Man-Month: Essays on Software Engineering (Anniversary Edition)*. Addison Wesley.

Browning, T. R. (2001). Applying the design structure matrix to system decomposition and integration problems: A review and new directions. *IEEE Transactions on Engineering Management,* Vol. 48, No. 3, pp. 292-306

Burton, R.M. and Obel, B. (1998). *Strategic Organizational Diagnosis and Design*. Kluwer Academic Publishers, Norwell, MA.

Cadiz, J.J., Venolia, G.D., Jancke, G., Gupta, A. (2002). Designing and deploying an information awareness interface. In *Proceedings of the Conference on Computer Supported Cooperative Work* (CSCW'02), New York, NY.

Carley, K.M. (2002). Smart Agents and Organizations of the Future. In *Handbook of New Media*. Edited by Lievrouw, L. and Livingstone, S., Sage, Thousand Oaks, CA.

Carley, K.M and Ren, Y. (2001). Tradeoffs between Performance and Adaptability for C^3I Architectures. *In Proceedings of the 6th International Command and Control Research and Technology Symposium*, Annapolis, Maryland.

Cataldo, M., Wagstrom, P, Herbsleb, J.D. and Carley, K.M (2006). Identification of Coordination Requirements: Implications for the Design of Collaboration and Awareness Tools. In *Proceedings of the Conference on Computer Supported Cooperative Work* (CSCW'06), Banff, Alberta, Canada.

Cataldo, M., Bass, M, Herbsleb, J.D. and Bass, L (2007). On Coordination Mechanism in Global Software Development. In *Proceedings of the International Conference on Global Software Engineering*, Munich, Germany.

Cataldo, M. and Nambiar, S. (2009). On the Relationship between Process Maturity and Geographic Distribution: an Empirical Analysis of their Impact of Software Quality. In *Proceedings of the International Conference on Foundations of Software Engineering (FSE'09),* Amsterdam, The Netherlands.

Cataldo, M. Mockus, A., Roberts, J.A., Herbsleb, J.D. (2009). Technical Dependencies, Work Dependencies and their Impact of Failures. *IEEE Transactions on Software Engineering*, Vol. 35, No. 6, pp. 864-878

Chidamber, S.R. and Kemerer, C.F. (1994). A Metrics Suite for Object-Oriented Design. *IEEE Transactions on Software Engineering*, Vol. 20, No. 6, pp. 476-493.

Clements, P., Bachman, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R. and Sttaford, J. (2002). *Documenting Software Architectures: Views and Beyond*. Addison-Wesley, New York, NY.

Conway, M.E. (1968). How do committees invent? *Datamation*, Vol. 14, No. 5, 28-31.

Curtis, B. (1981). *Human Factors in Software Development*. Ed. by Curtis, B., IEEE Computer Society.

Curtis, B., Kransner, H. and Iscoe, N. (1988). A field study of software design process for large systems. *Communications of ACM,* Vol. 31, No. 11, pp. 1268-1287.

de Souza, C.R.B., Redmiles, D., Cheng, L., Millen, D. and Patterson, J. (2004). How a Good Software Practice Thwarts Collaboration – The multiple roles of APIs in Software Development. In *Proceedings of the 12th Conference on Foundations of Software Engineering (FSE '04)*, Newport Beach, CA, 221-230.

de Souza, C.R.B. (2005). *On the Relationship between Software Dependencies and Coordination: Field Studies and Tool Support*. Ph.D. dissertation, Donald Bren School of Information and Computer Sciences, University of California, Irvine.

Ebert, C. and Jones, C. (2009). Embedded Software: Facts, Figures and Future. IEEE Computer, Vol. 42, No. 4, pp. 42-52.

Eclipse Project. (2009). http://www.eclipse.org. URL accessed on November 1st, 2009.

Eppinger, S.D., Whitney, D.E., Smith, R.P. and Gebala, D.A. (1994). A Model-Based Method for Organizing Tasks in Product Development. *Research in Engineering Design*, Vol. 6, pp. 1-13.

Espinosa, J.A. et al (2007). Familiarity, Complexity, and Team Performance in Geographically Distributed Software Development. *Organization Science*, Vol. 18, No. 4, pp. 613-630.

Faraj & Xiao (2006). Coordination in Fast-Response Organization. *Management Science*, Vol. 52, No. 8, pp. 1155-1169

Fenton, N.E. and Neil, M. (1999). A Critique of Software Defect Prediction Models. *IEEE Transactions on Software Engineering*, Vo. 25, No. 5, pp. 675-689.

Fisher, D., Brush, A.J., Gleave, E. and Smith M.A. (2006). Revisiting Whittaker and Sidner's "Email Overload": Ten Years Later. In *Proceedings of the Conference on Computer Supported Cooperative Work* (CSCW'06), Banff, Alberta, Canada.

Galbraith, J.R. (1973) *Designing Complex Organizations*. Addison-Wesley Publishing.

Gall, H. Hajek, K. and Jazayeri, M. (1998). Detection of Logical Coupling Based on Product Release History. In *Proceedings of the International Conference on Software Maintenance (ICSM '98)*, Bethesda, Maryland.

Garcia, A., et al. (2007). Assessment of Contemporary Modularization Techniques, ACOM'07 Workshop Report. *ACM SIGSOFT Software Engineering Notes*, Vol. 35, No. 5, pp. 31-37.

Geyer, W., Brownholtz, B., Muller, M., Dugan, C., Wilcox, E. and Millen, D.R. (2007). Malibu Personal Productivity Assistant. In *Proceedings International Conference on Human Factors in Computing Systems (CHI'07) – Work in Progress Section*, San Jose, CA.

Graves, T.L., Karr, A.F., Marron, J.S. and Siy, H. (2000). Predicting Fault Incidence Using Software Change History, *IEEE Transactions on Software Engineering*, Vol. 26, No. 7, pp. 653-661.

Gittel, J. H. (2002). Coordination Mechanisms in Care Provider Groups: Relational Coordination as a Mediator and Input Uncertainty as Moderator of Performance Effects. *Management Science*, Vol. 48, No. 11, pp. 1408-1426

Grinter, R.E., Herbsleb, J.D. and Perry, D.E. (1999). The Geography of Coordination Dealing with Distance in R&D Work. In *Proceedings of the Conference on Supporting Group Work (GROUP'99)*, Phoenix, Arizona.

Gokpinar, B., Hopp, W. and Iravani, S.M.R. (2009). The Impact of Misalignment of Organization Structure and Product Architecture on Quality of Complex Product Development. Forthcoming in Management Science.

Hassan, A.E. and Holt, R.C. (2004). C-REX: An Evolutionary Code Extractor for C. *CSER Meeting*. Canada, 2004

Henderson, R.M. and Clarck, K.B. (1990). Architectural Innovation: The Reconfiguration of Existing Product Technologies and the Failure of Established Firms. *Administrative Science Quarterly*, Vol. 35, pp. 9-30.

Herbsleb, J.D., Mockus, A., Finholt, T.A., and Grinter, R.E. (2000). Distance, Dependencies, and Delay in a Global Collaboration. *In Proceedings of the Conference on Computer Supported Cooperative Work* (CSCW'00), Philadelphia, Pennsylvania.

Herbsleb, J.D.and Moitra, D. (2001).Global Software Development. *IEEE Software*, March/April, pp. 16-20.

Herbsleb, J.D. and Mockus, A. (2003). An Empirical Study of Speed and Communication in Globally Distributed Software Development. *IEEE Transactions on Software Engineering,* Vol. 29, No. 6, pp.

Herbsleb, J.D., Mockus, A. and Roberts, J.A. (2006). Collaboration in Software Engineering Projects: A Theory of Coordination. In *Proceedings of the International Conference on Information Systems (ICIS '06)*, Milwaukee, Wisconsin.

Hoegl, M., Weinkauf, K. and Gemuenden, H. G. (2004). Interteam Coordination, Project Commitment and Teamwork in a Multiteam R&D Project: A Longitudinal Study. *Organization Science*, Vol. 15, No. 1, pp. 38-55

Horwitz, S., Reps, T., and Binkley, D. (1990). Interprocedural slicing using dependence graphs. *ACM Transactions on Programming Languages and Systems*, Vol. 22, No. 1, 26-60.

Hutchens, D.H. and Basili, V.R. (1985). System Structure Analysis: Clustering with Data Bindings. *IEEE Transactions on Software Engineering*, Vol. 11, No. 8, pp. 749-757.

Jazz Project (2009). http://jazz.net/pub/index.jsp. URL accessed on November 1st, 2009.

Karolak, D.W. (1998). *Global Software Development: Managing Virtual Teams and Environments*, IEEE Computer Society.

Kiczales, G. and Mezini, M.(2005). *Aspect-Oriented Programming and Modular Reasoning*.

Krackhardt, D. and Carley, K.M. (1998). A PCANS Model of Structure in Organization. In *Proceedings of the 1998 International Symposium on Command and Control Research and Technology*, pp.113-119.

Kraut, R.E. and Streeter, L.A. (1995). Coordination in Software Development. *Communications of ACM,* Vol. 38, No. 3, pp. 69-81.

Leffingwell, D. and Widrig, D. (2003). *Managing Software Requirements: A Use Case Approach, 2nd Edition*. Addison-Wesley.

Levchuk, G.M. et al. (2004). Normative Design of Project-Based Organizations – Part III: Modeling Congruent, Robust and Adaptive Organizations. *IEEE Trans. on Systems, Man & Cybernetics*, Vol. 34, No. 3, pp. 337-350.

Malone, T.W. and Crowston, K. (1994). The interdisciplinary study of coordination. *Comp. Surveys*, Vol. 26, No. 1, pp. 87-119.

March, J.G and Simon, H.A. (1958). *Organizations*. Wiley, New York, NY.

Mintzberg, H. (1979). *The Structuring of Organizations: A Synthesis of the Research*. Prentice-Hall, Englewood Cliffs, NJ.

Mockus, A., Fielding, R.T. and Herbsleb, J.D. (2002). Two Case Studies of Open Source Software Development: Apache and Mozilla. Transactions on Software Engineering and Methodology, Vol. 11, No. 3, pp. 309–346

Murphy, G.C., Notkin, D., Griswold, W.G. and Lan, E.S. (1998). An empirical study of call graph extractors. *ACM Transactions on Software Engineering Methodology*, Vol. 7, No. 2, pp. 158-191.

Nagappan, N and Ball, T (2007). Explaining Failures Using Software Dependencies and Churn Metrics. In *Proceedings of the 1st International Symposium on Empirical Software Engineering and Measurement*, Madrid, Spain.

Nagappan, N., Murphy, B., Basili, V.R. (2008). The Influence of Organizational Structure on Software Quality: An Empirical Case Study. In *Proceedings of the International Conference on Software Engineering (ICSE'08)*, pp. 521-530.

Northrop, L. (2006). *Ultra-Large-Scale System: The Software Challenge of the Future*. Software Engineering Institute, Carnegie Mellon University.

Olson, G.M. and Olson, J.S. (2000). Distance Matters. *Human-Computer Interaction*, Vol. 15, No. 2 & 3, pp. 139-178.

Parnas, D.L. (1972). On the criteria to be used in decomposing systems into modules. *Communications of ACM,* Vol. 15, No. 12, 1053-1058.

Paulk, M.C., Weber, C.V., Curtis, B. and Chrissis, M.B. (1995). *The Capability Maturity Model: Guidelines for Improving the Software Process*. Software Engineering Institute Series in Software Engineering, Addison-Wesley.

Ripley, R., Sarma, A. and van der Hoek, A. (2007). A Visualization for Software Project Awareness and Evolution. City, 2007

Sarma, A., Noroozi, Z. and van der Hoek, A. (2003). Palantir: Raising Awareness among Configuration Management Workspaces. In *Proceedings of the International Conference on Software Engineering (ICSE'03)*.

Sarma, A., Maccherone, L., Wagstrom, P. and Herbsleb, J. (2009). Tesseract: Interactive Visual Exploration of Socio-Technical Relationships in Software Development. In *Proceedings of the International Conference on Software Engineering (ICSE'09)*.

Selby, R.W. and Basili, V.R. (1991). Analyzing Error-Prone System Structure. *IEEE Transactions on Software Engineering*, Vol. 17, No. 2, pp. 141-152.

Schummer, T. and Haake, J.M. (2001). Supporting Distributed Software Development by Modes of Collaboration. In *Proceedings of the European Conference on Computer-Supported Collaborative Work (ECSCW '03*).

Shaw, M. and Garlan, D. (1996). *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall.

Simon, H.A. (1962). The Architecture of Complexity. *In Proceedings of the American Philosophical Society*, Vol. 106, No. 6, pp. 467-482.

Sosa, M.E., Eppinger, S.D., and Rowles, C.M. (2004). The Misalignment of Product Architecture and Organizational Structure in Complex Product Development. *Management Science*, Vol. 50, No. 12, pp. 1674-1689

Sosa, M. (2008). A Structured Approach to Predicting and Managing Technical Interactions in Software Development. *Research Engineering Design*, Vol. 19, pp. 47-70.

Staudenmayer, N. (1997). *Managing Multiple interdependencies in Large Scale Software Development Projects*. Unpublished Ph.D. Dissertation, Sloan School of Management, Massachusetts Institute of Technology,

Sullivan, K.J., Griswold, W.G., Cai, Y, and Hallen, B. (2001). The Structure and Value of Modularity in Software Design. *In Proceedings of the International Conference on Foundations of Software Engineering* (FSE '01), Vienna, Austria, 99-108.

Thompson, J.D. (1967). *Organizations in Action: Social Science Bases of Administrative Theory*. McGraw-Hill, New York, NY.

Trainer, E., Quirk, S., de Souza, C. and Redmiles, D. (2005). Bridging the Gap between Technical and Social Dependencies with Ariadne. In *Proceedings of Workshop on the Eclipse Technology Exchange*, San Diego, California.

von Hippel, E. (1990). Task Partitioning: An Innovation Process Variable. *Research Policy*, Vol. 19, pp. 407-418.

Wattenberg, M., Rohall, S., Gruen, D. and Kerr, B. (2005). E-Mail Research: Targeting the Enterprise. *Journal of Human-Computer Interaction*, Vol. 20, pp. 139-162.

Watts, D.J. (1999). *Small Worlds: The Dynamics of Networks between Order and Randomness*, Princeton University Press, Princeton, NJ.

Zimmerman, T. and Weibgerber, P. (2004). Preprocessing CVS Data for Fine-grain Analysis. In *Proceedings of the 1st International Workshop on Mining Software Repositories*, Edinburgh, Scotland, U.K.

Zimmerman, T., & Nagappan, N. (2008). Predicting Defects using Network Analysis on Dependency Graphs. In *Proceedings of the International Conference on Software Engineering (ICSE'08)*, 2008, pp. 531-540.

APPENDIX

TABLE A.1
DESCRIPTIVE STATISTICS OF VARIABLES USED IN DEVELOPMENT PRODUCTIVITY STUDY

PROJECT A: DISTRIBUTED SYSTEM						
	Mean	**SD**	**Min**	**Max**	**Skew**	**Kurtosis**
Resolution Time (log)	3.260	1.236	0	6.490	-0.809	3.127
Change Sixe (log)	1.163	1.781	0.301	4.741	0.302	4.005
Team Load	9.104	2.938	1.016	58.800	-0.361	2.342
Multiple Locations	0.779	0.414	0	1	-1.346	2.814
Shared Work Experience (log)	0.011	0.033	0	0.381	1.147	3.644
Component Experience (log)	3.051	0.958	0	5.601	-0.015	2.145
Structural Congruence (Logical Dep.)	0.663	0.217	0.156	0.995	-0.931	3.754
Geographical Congruence (Logical Dep.)	0.684	0.237	0.142	0.993	-0.863	3.201
MR Congruence (Logical Dep.)	0.567	0.283	0.070	0.982	-0.319	1.965
IRC Congruence (Logical Dep.)	0.599	0.274	0.079	0.982	-0.506	2.233
PROJECT B: EMBEDDED SYSTEM						
	Mean	**SD**	**Min**	**Max**	**Skew**	**Kurtosis**
Resolution Time (log)	3.384	0.623	0	4.025	-0.323	1.957
Change Sixe (log)	1.671	0.665	0.301	4.276	1.563	3.014
Team Load	9.098	2.923	1.016	13.992	-0.371	2.354
Multiple Locations	0.615	0.486	0	1	-0.475	1.226
Shared Work Experience (log)	0.084	0.072	0	0.540	1.629	2.465
Component Experience (log)	4.069	1.239	0	4.995	0.031	2.227
Structural Congruence (Logical Dep.)	0.625	0.178	0.009	0.990	-0.355	2.512
Geographical Congruence (Logical Dep.)	0.631	0.192	0.011	0.987	-0.451	2.687

TABLE A.2
DESCRIPTIVE STATISTICS OF VARIABLES USED IN SOFTWARE QUALITY STUDY

PROJECT A: DISTRIBUTED SYSTEM						
	Mean	**SD**	**Min**	**Max**	**Skew**	**Kurtosis**
File Buggyness	0.458	0.498	0	1	0.167	1.028
LOC (log)	5.097	1.792	0	9.789	1.191	1.517
Avg. Lines Changed (log)	1.389	1.316	0	6.511	1.871	2.359
Num. Logical Dep. (log)	3.789	1.641	0	6.729	0.655	1.911
Clustering Logical Dep. (log)	0.545	0.182	0	0.693	1.012	1.939
Coordination Req. (log)	0.035	0.053	0	0.215	-0.362	1.943
Structural Congruence (Logical Dep.)	0.268	0.129	0.156	0.421	0.284	1.089
Geographical Congruence (Logical Dep.)	0.432	0.064	0.142	0.476	1.090	1.517
MR Congruence (Logical Dep.)	0.247	0.336	0.070	0.982	1.606	2.359
IRC Congruence (Logical Dep.)	0.205	0.101	0.079	0.386	0.241	1.744
PROJECT B: EMBEDDED SYSTEM						
	Mean	**SD**	**Min**	**Max**	**Skew**	**Kurtosis**
File Buggyness	0.176	0.381	0	1	1.096	2.888
LOC (log)	2.793	1.307	0	8.587	0.878	1.318
Avg. Lines Changed (log)	1.944	1.058	0	8.861	1.264	1.757
Num. Logical Dep. (log)	0.417	0.697	0	4.111	1.710	3.345
Clustering Logical Dep. (log)	0.394	0.076	0	0.641	-2.216	2.234
Coordination Req. (log)	0.027	0.067	0	0.555	1.371	1.467
Structural Congruence (Logical Dep.)	0.093	0.164	0.009	0.891	1.578	2.950
Geographical Congruence (Logical Dep.)	0.219	0.147	0.012	0.477	0.110	1.753

TABLE A.3:
PAIR-WISE CORRELATIONS (* p < 0.01) AMONG VARIABLES USED IN DEVELOPMENT PRODUCTIVITY STUDY

PROJECT A: DISTRIBUTED SYSTEM

	1	**2**	**3**	**4**	**5**	**6**
1.*Change Sixe (log)*	-					
2.*Team Load*	-0.04	-				
3.*Multiple Locations*	0.01	0.01	-			
4.*Shared Work Experience (log)*	0.06	0.03	-0.01	-		
5.*Component Experience (log)*	0.13*	0.02	0.04	0.17*	-	
6.*Structural Congruence (Logical)*	0.04	0.03	0.05	-0.02	-0.05	-
7.*Geographical Congruence (Logical)*	0.05	-0.09	0.09	-0.01	0.01	0.13*
8.*MR Congruence (Logical)*	-0.01	-0.06	-0.04	-0.01	-0.01	0.03
9.*IRC Congruence (Logical)*	-0.02	-0.04	-0.03	-0.01	-0.02	0.03

	7	**8**	**9**	**10**	**11**	**12**
8.*MR Congruence (Logical)*	0.02	-				
9.*IRC Congruence (Logical)*	0.01	0.01	-			

PROJECT B: EMBEDDED SYSTEM

	1	**2**	**3**	**4**	**5**	**6**
1.*Change Sixe (log)*	-					
2.*Team Load*	0.09	-				
3.*Multiple Locations*	0.13*	0.01	-			
4.*Shared Work Experience (log)*	0.02	0.03	0.16*	-		
5.*Component Experience (log)*	0.26*	0.12*	0.03	0.33*	-	
6.*Structural Congruence (Logical)*	0.01	0.03	0.02	0.01	-0.01	-
7.*Geographical Congruence (Logical)*	0.02	0.02	0.02	0.02	-0.03	0.21*

TABLE A.4:
PAIR-WISE CORRELATIONS (* p < 0.01) AMONG VARIABLES USED IN SOFTWARE QUALITY STUDY

PROJECT A: DISTRIBUTED SYSTEM

	1	**2**	**3**	**4**	**5**	**6**
1.*LOC (log)*	-					
2. *Avg. Lines Changed (log)*	0.31*	-				
3.*Num Logical Dep. (log)*	0.11*	0.43*	-			
4.*Clustering Logical Dep. (log)*	0.31*	0.34*	0.06*	-		
5.*Coordination Req. (log)*	-0.32*	-0.21*	-0.10*	-0.14*	-	
6.*Struct. Congruence (Logical)*	0.33*	0.07*	-0.05	-0.03	0.31*	-
7.*Geo. Congruence (Logical)*	0.04*	-0.04	-0.04	-0.01	0.12*	0.08*
8.*MR Congruence (Logical)*	0.19*	0.09*	-0.01	0.03	0.11*	0.09*
9.*IRC Congruence (Logical)*	0.02	-0.06*	-0.01	0.05	0.02	0.11*

	7	**8**
8.*MR Congruence (Logical)*	0.11*	-
9.*IRC Congruence (Logical)*	0.05	0.17*

PROJECT B: EMBEDDED SYSTEM

	1	**2**	**3**	**4**	**5**	**6**
1.*LOC (log)*	-					
2. *Avg. Lines Changed (log)*	0.27*	-				
3.*Num Logical Dep. (log)*	0.11*	0.41*	-			
4.*Clustering Logical Dep. (log)*	0.21*	0.27*	0.16*	-		
5.*Coordination Req. (log)*	-0.18*	-0.11*	-0.19*	0.04	-	
6.*Struct. Congruence (Logical)*	0.33*	0.05	0.03	0.03	0.11*	-
7.*Geo. Congruence (Logical)*	0.04*	0.02	0.02	0.01	0.04	0.07*