Coordination Breakdowns and Their Impact on Development Productivity and Software Failures

Marcelo Cataldo and James D. Herbsleb

Abstract—The success of software development projects depends on carefully 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. Although that research stream has been quite influential, it considers a static view of the problem of coordination in engineering activities. Building on a dynamic view of coordination, we studied the relationship between socio-technical congruence and software quality and development productivity. In order to investigate the generality of our findings, our analyses were performed on two large-scale projects from two companies with distinct characteristics in terms of product and process maturity. Our results revealed that the gaps between coordination requirements and the actual coordination activities carried out by the developers significantly increased software failures. Our analyses also showed that higher levels of congruence are associated with improved development productivity. Finally, our results showed the congruence between dependencies and coordinative actions is critical both in mature development settings as well as in novel and dynamic development contexts.

Index Terms—Metrics/measurement, productivity, organizational management and coordination, quality analysis and evaluation

1 Introduction

Over the past decades, software-intensive systems have become increasingly pervasive in our lives, and their complexity has increased drastically [1]. 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 the distribution of the development work and a corresponding increase in organizational complexity [2], [3], [4]. Coordination—long recognized as one of the fundamental problems of software engineering [5], [6]—has become ever more challenging. This has led to a growing body of work on coordination in software development (e.g., [7], [8], [9], [10]).

Coordination is a topic that has also received significant attention in the product development and organizational theory literatures. Both research streams have proposed similar theoretical perspectives based on the idea of "nearly decomposable" systems [11], [12], [13], [14]. On the technical dimension, the work on modular product designs has extensively examined the role of interdependencies among components of a product and has proposed approaches to minimize those dependencies [15], [16], [17]. A key assumption in this line of work is that minimizing technical dependencies among product components will result in a modular work structure [15], [18]. In a similar vein, organizational researchers have presented arguments from the perspective of organizing the work around loosely coupled entities such as teams or departments [19], [20], [21], [22].

Despite their contributions, 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 [23]. Moreover, modular structures as well as traditional organizational mechanisms for coordination tend not to be suitable for environments with volatile dependencies [7], [12], [24]. In other words, these well-established theoretical perspectives have taken a static view of the problem of coordination in engineering activities, tacitly assuming that product structure and organizational structure are established early and do not change [25]. For this reason, they fail to fully recognize the complex and dynamic reality of software development projects. More recently, organizational researchers have argued that informal coordination mechanisms such as direct interaction among project members are very valuable in highly interdependent task contexts [12], [24]. Unfortunately, those approaches fail to adequately manage the scale of large projects and rely on the individuals' ability to identify the relevant dependencies, which is a constant challenge in software development (see, for instance, [26], [8], [10], [27]).

This paper examines the impact of adequately identifying and managing coordination needs on two fundamental software development outcomes: development productivity

- *M. Cataldo is with the Research and Technology Center, Robert Bosch LLC, 2835 East Carson Street, #210 Pittsburgh, PA 15203. E-mail: marcelo.cataldo@us.bosch.com.*
- *J.D. Herbsleb is with the School of Computer Science, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213. E-mail: jdh@cs.cmu.edu.*

Manuscript received 17 Mar. 2010; revised 17 Oct. 2010; accepted 28 Apr. 2012; published online 23 May 2012.
Recommended for acceptance by M.-A. Storey.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-2010-03-0079.
Digital Object Identifier no. 10.1109/TSE.2012.32.

and software quality. In previous work [28], [7], we, together with our collaborators, proposed a new perspective on coordination that allows a more adequate consideration of dynamic dependencies. Our results showed that when coordination needs are matched by appropriate coordinating actions, a state defined as socio-technical congruence [28], development productivity improves. In addition, our past work showed that coordination needs could be quite volatile during a software project [7]. The work reported in this paper extends that earlier empirical work in three complementary ways. First, our empirical analyses showed that higher levels of congruence are associated with a significant reduction in software failures results. Second, we replicated our original results [28] on productivity in a second large-scale project. This new research setting differs significantly in terms of product, process, and organizational maturity from the organization studied in our original work [28]. Finally, we report new insights on the nature of the volatility of dependencies when coordination needs cut across organizational and geographical boundaries.

The rest of the paper is organized as follows: We first discuss the traditional perspectives on dependencies and coordination. Second, we build on our past work on socio-technical congruence concept to articulate the research questions examined in this paper. Third, we describe our research setting followed by our empirical analyses and results. We conclude with a discussion of the implications of our results and future research directions.

2 Dependencies in Software Development: The Traditional Perspective

Over the years, work dependencies and coordination have received significant attention from product development and organizational researchers. On the organizational dimension, numerous approaches to manage the work dependencies that span organizational entities such as individuals or teams have been proposed. March and Simon [21] argued that schedules and feedback mechanisms are required when interdependence is unavoidable. Thompson [22] 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 [19] argued that low levels of interdependency could be managed by traditional mechanisms such as rules, plans, and processes. However, as the level of interdependency increases, additional mechanisms such as lateral communication are required [19]. Malone and Crowston [20] developed a typology of coordination problems to catalog coordination mechanisms that address specific types of interdependencies. The formal coordination mechanisms proposed by the organizational theory literature are appropriate in predictable and stable settings where a priori definition of organizational structures and processes for managing dependencies are feasible [12], [29]. However, formalization has important limitations and they tend to be overcome by the emergence of an informal organization where interpersonal relationships play a central role [30].

Recent work has shown that interpersonal coordination mechanisms are better suited for managing highly interdependent tasks because the information processing capabilities of interactions complement traditional coordination mechanisms [24]. However, in large-scale software projects characterized by rapid changes, geographic dispersion, and multiple organizational boundaries, these types of informal coordination mechanisms could be prohibitively costly [31], [9], [32]. For instance, identifying and managing relevant technical and work dependencies across geographic boundaries is quite challenging [32], impacting productivity [28], [9] and software quality [31].

In the product development literature, traditional approaches to coordination are based on the decomposability view (see [11] and [33] for reviews). That theoretical perspective argues that complexity, technical or organizational, can be managed by dividing a system or a task into smaller and more manageable units [13], [34]. In this context, the work is often characterized as choosing the design parameters for a system [15], [35]. Such decisions are often highly interrelated, 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 [36], are made first and establish the stable "design rules" that determine how components will interact [15]. 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 entities [16], [18], [34] to work in parallel on different components.

A modular approach to reducing technical and task dependencies, however, is not adequate under conditions where the product decomposition and interfaces are uncertain and constantly changing. This is particularly relevant in software engineering, where it is widely accepted that the requirements of the system become known gradually over time, and requirements change as time progresses [5], [37]. In some cases, the changes in the requirements result in minor modifications in the system. In other cases, new features have to be added or existing features are eliminated. Those changes can 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. Furthermore, teams in a modularized system tend to communicate relatively infrequently, making it harder to achieve mutual understanding when the decomposition or interface does change [26], [10]. Moreover, "cross-cutting" product features pose challenges for traditional modular designs because the changes across multiple modules need to be coordinated [23].

3 Dependencies and Their Implications for Software Failures and Development Productivity

We, together with our collaborators, introduced the concept of socio-technical congruence as a new perspective on coordination where evolving dependencies are front and center [28], [7]. In that conceptualization, socio-technical congruence is defined as the degree of matching between two elements, coordination requirements and actual coordination activities. Traditional coordination perspectives have worked quite well in stable settings. In such context, coordination requirements are typically articulated as high-level and stylized categorizations of dependencies (e.g., task A needs to be completed before task B can start). However, those perspectives suffer from serious limitations in rapidly changing work contexts [12]. In contrast, our congruence approach considers task dependencies as dynamic relational entities that can be defined at any level of granularity. For instance, we showed that logical dependencies are a more reliable predictor of coordination requirements than traditional syntactic dependencies [28], presumably because they are better able to capture semantic relationships among code elements.

Gaining a deeper understanding of the role of socio-technical congruence in software development involves examining the relationship between congruence and critical outcomes such as software quality and development productivity. Furthermore, generalizing those results requires understanding whether that relationship remains consistent across settings with different contextual factors, such as the novelty of a product. The following paragraphs discuss the hypothesized relationship between congruence and software failures and productivity.

3.1 Congruence and Software Failures

Mismatches between coordination requirements derived and coordination behavior have been shown to have a negative impact on the quality of complex systems such as airplane engines [38] or automobiles [39]. In software engineering, coordination breakdowns can lead to higher number of defects and higher costs [5], [40]. An important challenge in software engineering is that software failures can stem from various types of dependencies, which in many cases are not easily identifiable (e.g., [41]). Furthermore, traditional design techniques such as modularization sometimes have undesirable side effects. For example, several researches have documented how the use of modular designs to reduce technical dependencies tends to lead development teams to assume an exaggerated degree of independence (e.g., [8], [10]). The end result is a significantly higher number of integration problems. Therefore, we expect that identifying the relevant dependencies, and the corresponding coordination needs, combined with adequate coordination actions should result in a reduction in software failures. This leads to our first research question:

RQ 1: Are higher levels of socio-technical congruence associated with lower levels of software failure?

The nature of the relationship between software dependencies, coordination requirements, and software quality could be impacted by at least two important contextual factors: product maturity and process maturity. Early in the development life cycle of a new software system, coordination needs tend to be very dynamic. The development organization needs to grasp the novelty of the system to be built, learn and understand the requirements, and define and realize the structure of the system [42], [7], [5], [43]. In this context, a detailed view of coordination can be very valuable to better understand how such dynamism impacts the quality of a software system. As systems mature, the nature of the coordination needs changes. For example, changes to the system tend to face more scrutiny and inertia [14], suggesting less dynamism. However, modifications in mature systems are likely to involve or impact a higher number of components or modules than in less mature systems [44]. For these reasons, the coordination costs, the likelihood of coordination breakdowns, and, consequently, the likelihood of software failures associated with those changes could increase. These opposing forces suggest that the relationship between socio-technical congruence and software failures should be examined in different product maturity contexts.

A second dimension to consider is the maturity in the development processes. As development organizations achieve higher levels of maturity in their processes, the quality of the systems produced by such organizations tends to increase [45], [46]. Past research has shown that standardized processes are very valuable when dependencies among tasks are known a priori and stable. However, standardized processes as a coordination mechanism have major limitations when coordination needs are dynamic, uncertain, and difficult to identify [47], [12], [19], [29]. Those limitations may also apply even in the case of organizations with mature processes. This leads to our second research question:

RQ 2: Does the relationship between socio-technical congruence and software failure persist across mature and less mature development contexts?

3.2 Congruence and Software Development Productivity

The organizational literature suggests that congruence is an important factor affecting task performance (e.g., [48], [49]). In software development, coordination breakdowns can lead to longer development times [50], [9]. Our earlier work [28], [7] showed that congruence was associated with improvements in the resolution time of development tasks in a large-scale project that developed a novel software system. The previous section argued that the relationship between congruence and software quality could differ across mature or less mature development organizations as well as across products that are novel or mature. The basis of such an argument is also applicable to development productivity. This leads to the last two research questions addressed in this paper:

RQ 3: Are higher levels of socio-technical congruence associated with higher levels of development productivity?

RQ 4: Does the relationship between socio-technical congruence and development productivity persist across mature and less mature development contexts?

4 Research Setting

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

4.1 Project A: Distributed System

Project A was a large distributed system for a data storage product. The data covered 39 months of development activity corresponding to the first four releases of the product. The company had been in existence for 21 months prior to the beginning of the period covered by our data. The company had 114 developers grouped into eight development teams distributed across three development locations in North America. All the members of each team were colocated. 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 7,737 source code files, mostly in C language, and a small portion (117 files and less than 96,000 lines of code) in C++ language. The system consisted of 27 architectural components and the development responsibility of each one was assigned to only one development team. All developers had full access to a Perforce version control system and a modification request (MR) tracking system based on the Bugzilla open source system. The development organization had a collection of scripts that automated the linking of commits to the version control system with the MR tracking system. A modification request represents a unit of work such as a defect, an enhancement to an existing feature, or the development of a new feature. The data corresponded to a total of 8,257 resolved 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 impromptu interactions. Developers also used tools such as Internet Relay Chat (IRC) and an 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.

4.2 Project B: Embedded System

We collected data from a multinational 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 the end of 2008. Three hundred eighty developers located in eight development sites distributed across Europe and India participated in the project. Those developers were organized in 37 development teams. Thirteen developers had been with the development organization for less than one year and all developers underwent training related to all the tools and processes used in the development projects. The system was composed of about 7 million lines of code distributed in 9,074 source code files and 562 architectural components. The development responsibilities of each component were assigned to a single development team. All source code files were written in the C language. All developers had full access to the version control system and the modification request tracking system. This development organization used ClearCase and ClearQuest as the version control system and MR tracking system, respectively. Both systems were highly customized to the development processes of the organization and a collection of scripts was used to link an MR with the revisions of the impacted source code files through the use of labels in the version control system. The data corresponding to a total of 4,170 resolved modification requests were identified and 2,340 of those were multiteam modification requests. The multiteam MRs involved 10,736 commits to the version control system.

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

4.3 Comparison across Projects

Research questions 2 and 4 focus on the relationship between socio-technical congruence and development outcomes in development contexts that differ in maturity. The two projects studied were chosen because they represent two very distinct points, differing on both of two dimensions of maturity: product maturity and process maturity. Table 1 highlights the key differences where project A represents the less mature development context while project B exhibited high levels of product and process maturity.[1] It is important to notice that the two dimensions of maturity, product and process maturity, are not completely orthogonal. In fact, they might be confounded. Such a limitation can be addressed from an experimental design point of view. If the impact of congruence is the same across the two projects, then the confounding effect between product maturity and process maturity does not pose a problem. On the other hand, the confounding represents a challenge if the relationship between congruence and the outcome variables, software quality and development productivity, is not consistent across projects. In this case, the confounding between product and process maturity will not allow us to disentangle which of the factors is influencing the results.

5 Computing Coordination Requirements and Congruence

In this section, we describe how the coordination requirements and congruence measures were computed from the collected data. Our socio-technical congruence approach [28], [7] provides us with a measure of coordination requirements

1. The data collected from both projects also points out a difference in terms of what amount of development work each MR represented. Our analyses as well as data collected in interviews revealed project-specific characteristics that we think drove those differences. Members in project B tended to define tasks as units of work that would take at least one or more weeks worth of work. Labor regulations in European countries were the main driver behind such an approach. In the case of project A, project members tended to define tasks based solely on the technical aspects. Such a difference is evidenced in the significantly higher variance in the resolution time on project A's tasks.

TABLE 1
Differences between Projects along Product and Process Dimensions of Maturity

	Project A	Project B
Product Maturity	Novel Product First in its class, based on an academic project	Market-leader product 8th generation of the product
Process Maturity	A combination of agile and open source processes (e.g. daily builds, minimal focus on requirements engineering)	Strong focus on processes. Well-established use of the V-model methodology.
	No CMM assessment	Two locations were assessed CMM Level 5, five locations were CMM Level 3 and the remaining one was CMM Level 2
	Relatively immature development organization. Our data cover 39 of the first 60 months of the company existence	Very matured development organization with more than 20 years developing the various generations of the system

which is computed from two elements: a people-to-task relationship (task assignment matrix—T_A) and a task-to-task relationship (task dependency matrix—T_D). In our research settings, an MR represents a unit of development work where one or more engineers can be involved. The work performed by those individuals in an MR is materialized in changes to software artifacts such as source code files. Those changes may impact other source code files or might be impacted by changes or properties of other files. The technical dependencies among source code files represent a key relationship that allows us to examine or assess the impact or influence of changes among source code files. Then, the potential impact of a change represents a potential need to coordination among developers working within a given MR or between MRs. This set of linkages between individuals, MRs, and software artifacts such as source code files allows us to construct T_A and T_D.

More specifically, the necessary data can be extracted from two types of repositories commonly encountered in software development projects: the MR tracking systems and the version control systems. An MR provides the "developer i modified file j" relationship that constitutes our T_A matrix. The task dependency (T_D) matrix can be articulated as the technical dependencies among the different source code files. Our past work showed that logical dependencies[2] were a significantly better predictor of coordination needs than syntactic dependencies [28]. Therefore, we considered logical dependencies for our analyses. In that case, the cell ij in 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 MRs prior to the focal MR. Finally, the coordination needs matrix C_R is computed as $T_A * T_D * transpose(T_A)$ and the result a people to people matrix where each cell ij indicates the extent to which the work of developers i and j is interdependent.[3]

Computing congruence involves comparing the C_R matrix against another person-to-person matrix, called actual coordination matrix C_A. The C_A captures the coordination activities performed by the individuals working on development tasks [28], [7]. As described in our past work [28], [7], congruence is formally defined as the quotient between the number of cells ij in C_R and C_A that are nonzero and the number of cells ij in C_R that are nonzero.

Development organizations may coordinate work in different ways; therefore, we constructed two distinct C_A matrices for each project. One C_A matrix captured the existence of coordination activity between engineers i and j if they belonged 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 perfectly modular, then the work in one module can be done within one team and no coordination is required among different teams. Relating this matrix, $C_{A\text{-}Struct}$, matrix to the C_R matrix gives us the *Structural Congruence* measure.

Similarly, a second matrix $C_{A\text{-}Geo}$, was built around the idea of potential paths of communication and coordination that exist when individuals work in the same physical location [51], [32]. In terms of the matrix of coordination activities, engineers i and j have a linkage if they work in the same location. Relating the $C_{A\text{-}Geo}$ matrix to the C_R matrix gives us the *Geographical Congruence* measure. Fig. 1 describes an example of computing the congruence measures. In the case of structural congruence, all three

$$\text{Structural Congruence} = \text{Diff}\left(\overset{C_R}{\begin{bmatrix} - & 1 & 10 \\ 1 & - & 4 \\ 10 & 4 & - \end{bmatrix}}, \overset{C_{A\text{-Strut}}}{\begin{bmatrix} - & 1 & 1 \\ 1 & - & 1 \\ 1 & 1 & - \end{bmatrix}} \right) / 6 = 1$$

$$\text{Geographical Congruence} = \text{Diff}\left(\overset{C_R}{\begin{bmatrix} - & 1 & 10 \\ 1 & - & 4 \\ 10 & 4 & - \end{bmatrix}}, \overset{C_{A\text{-Geo}}}{\begin{bmatrix} - & 1 & 1 \\ 1 & - & 0 \\ 1 & 0 & - \end{bmatrix}} \right) / 6 = 0.66$$

$$\text{MR Congruence} = \text{Diff}\left(\overset{C_R}{\begin{bmatrix} - & 1 & 10 \\ 1 & - & 4 \\ 10 & 4 & - \end{bmatrix}}, \overset{C_{A\text{-MR}}}{\begin{bmatrix} - & 3 & 3 \\ 3 & - & 0 \\ 3 & 0 & - \end{bmatrix}} \right) / 6 = 0.66$$

$$\text{IRC Congruence} = \text{Diff}\left(\overset{C_R}{\begin{bmatrix} - & 1 & 10 \\ 1 & - & 4 \\ 10 & 4 & - \end{bmatrix}} \quad \overset{C_{A\text{-IRC}}}{\begin{matrix} - & 2 & 5 \\ 2 & - & 3 \\ 5 & 3 & - \end{matrix}} \right) / 6 = 1$$

Fig. 1. Example of congruence computation.

2. The idea of logical dependencies was first articulated by Gokpinar et al. [39] as files changed together as part of a unit of work.
3. A step-by-step example of the calculation of the CR matrix can be found in our previous work [28].

developers belong to the same team, so the $C_{A\text{-}Struct}$ matrix contains all 1s, resulting in a congruence measure of 1 since the C_R matrix indicates that all three developers have some level of interdependence (off-diagonal cells are nonzero). On the other hand, the information reported in the $C_{A\text{-}Geo}$ matrix indicates that one of the three developers is not in the same geographical location, resulting in a value of geographical congruence of 0.66.

In the case of project A, we had data associated with two additional communication and coordination mechanisms: the exchanges of information in the MR tracking system and IRC. Using such data, we constructed two additional measures of congruence. *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. All duplicates of the focal MR were also used to capture the interactions among developers. We focused on interactions among developers that explicitly commented on the MR report and we did not consider those individuals that just received notification e-mails every time an MR is updated. The creator of the MR was included in the construction of the MR Communication Congruence measure. *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. They then 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 percent of the MRs were coded by all raters. Comparisons of the obtained networks showed that 98.2 percent of the networks had the same set of nodes and edges. In Fig. 1, the $C_{A\text{-}MR}$ and $C_{A\text{-}IRC}$ matrices have cells with values higher than one because those cells represent the number of times the pair of individuals interacted through that particular communication means.

6 Socio-Technical Congruence and Software Failures

In this section, we present our empirical examination of the relationship between congruence and software failures across our two distinct development projects (RQ1 and RQ2). The contents are organized as follows: We first describe the various measures used in our analyses, followed by a description of the statistical model. We conclude with a presentation of the results.

6.1 Description of the Measures

We considered software failures as failure proneness, which is defined as the likelihood of software entities (e.g., source code files) being modified as part of fixing a field defect. The literature has identified a number of factors that impact failure proneness (e.g., [41], [52], [53]). 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 tasks. We collected measures across all three of those dimensions and the following paragraphs describe them.

6.1.1 Measuring Failure Proneness

Our dependent variable, *File Buggyness*, is a binary measure indicating whether a file has been modified as part of resolving a field defect. Therefore, the unit of analysis is the source code file. We selected this particular way of examining software quality for two reasons. First, the measure has been empirically evaluated in prior research (see, for instance, [40] and [41]). Second, examining the linkage between software failures and the related source code files has very important practical implications [41].

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 postrelease defects are quite severe because they typically involve a recall of vehicles with significant financial and reputational impact. Great effort goes into quality assurance and, consequently, postrelease defects seldom occur. Therefore, we considered "field defects" as those defects encountered during the integration and system-testing phase.[4] 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 it was associated with a field defect in any of the releases of the product covered by the data. However, considering only those files that were modified by the development activities can result in a selection bias. For example, defects might exist in source code files that were not modified as part of development activities in a particular time period and they are discovered at some point in time after a release of the product. Therefore, we also included all files that were associated with field defects that did not change during the development of a release under study.

6.1.2 Congruence Measures

In order to construct the congruence measures, we used the same formulation proposed in our earlier work [28], [7]. However, the outcome of such computation is a measure of congruence per modification request. Since our unit of analysis is the source code file, 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. Subsequently, 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, such a measure was highly correlated with our control factors. Hence, it was more appropriate to use the measure based on the median value of congruence. Measures based on structural and geographical congruence were computed for both projects, while IRC and MR congruence measures were computed only for project A.

4. We recognize that equating defects found in an integration and system testing phase with field defects may be inappropriate for some purposes. However, we think that in the context of project B, it is not an issue for concern because system testing involves evaluating the software system in a real automobile and, due to safety concerns, reaching this point requires the quality of the system to be in many respects as high as those found in released product in other industries.

TABLE 2
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 χ2	1663**	1701**	3092**	3401**
df	5	7	5	7
Deviance Explained	0.302	0.317	0.362	0.401
Model Comparison χ2	--	38.13**	--	309.07**

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

6.1.3 *Additional Control Factors*

Past research on failure proneness has identified a number of technical and work-related factors that impact it (e.g., [54], [55], [52], [53]). Building on such a line of work, we examined, together with our collaborators [41], the relationship of failure proneness with several process, product, and work-related metrics in two large-scale industrial software development projects from two distinct companies. We identified a subset of technical and work dependencies related factors that had consistent impact across both projects. Given the demonstrated generality of the role of those factors in the context of failure proneness, we decided to consider them as the set of control variables used in this study. The subsequent paragraph describes those measures.

The *size of the file (LOC)* was computed as the number of nonblank noncomment lines of code. We also computed the *average number of lines changed* in a file as part of the modification requests. Using the information in the T_D matrices described in Section 5, we computed the two technical dependency measures as proposed in our earlier work [41]. *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 by 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. A "neighbor of file i" is a file that has logical dependencies with file i. This measure is equivalent to Watts's [56] local clustering measure. We computed the *Coordination Needs Dependencies* measure, which captures for each file i, the degree centrality of the most central developer in the C_R matrix that change such file [41].

6.2 Description of the Statistical Model

Since our dependent measure is a binary variable, our analyses use the following logistic regression model to assess the impact of socio-technical congruence on failure proneness:

$$P(Buggyness = 1|X) = \left[1 + e^{\sum_i \alpha_i * CongruenceMeasure_i + \sum_k \beta_k * ControlMeasure_k}\right]^{-1},$$

where X is the vector consisting of the congruence and control measures. As 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. In particular, we report the 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.

We report odds ratios instead of the traditional regression coefficients 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 *size of the file* 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. Finally, it is worth noting that the data for project A covers four releases of the product; therefore, the source code files can have multiple data points in the dataset. In order to adequately handle the repeated measures and the random effects associated with idiosyncrasies of each release, we used a multilevel logistical regression model.

6.3 Results

We performed several logistic regression analyses to assess the effect of the congruence measures on failure proneness of source code files. Table 2 reports the odds ratios associated with the various regression models. We constructed a baseline model for both project A and B (models I and III) considering only the control factors. The results from both projects are consistent in terms of the directionality and magnitude of the effects with those reported in our past work [41]. The odds ratios associated with the clustering of logical dependency measure are quite small. This suggests that increases in this measure result in a significant reduction in the likelihood of failure proneness of a source code file. Specifically, we consider ceteris paribus, the change in the outcome variable between the minimum and maximum observed values of the clustering measure. Fixing the other independent variables at their means in project A; for instance, we observe a 71.6 percent reduction in the estimated probability of a source code file. The estimated probability goes from 0.88 (for the minimum observed value of the clustering measure) down to 0.25 (for the maximum observed value). In the case of project B, we see a 90.4 percent reduction in the estimated probability of a file being associated with a field defect.

Models II and IV introduce the measures of congruence based on logical dependencies for projects A and B, respectively. The results show a statistically significant impact of both structural and geographical congruence in project B (model IV). Higher levels of congruence are associated with lower levels of failure proneness (odds ratios less than 1). In project A, only structural congruence was statistically significant. The magnitude of the impact of congruence differs across projects, as evidenced by the substantially smaller values in the odds ratios in project A relative to project B. In the less mature development setting, such project A improvements in the levels of congruence seem to be associated with larger reduction of the likelihood of software failures (smaller odds ratios) than in the more mature setting such as project B (odds ratios closer to 1). We can quantify those differences by considering the minimum and maximum observed values of the congruence and calculating the change in the estimated probability of failure associated with the source code files while fixing the other variables at their means. For instance in project A, a change in structural congruence from 0.156 (the minimum observed value) to 0.421 (the maximum observed value) results in a reduction of 18.6 percent in the estimated probability of failure associated with a source code file. In project B, a change in structural congruence from 0.009 to 0.891 results in a reduction of 7.6 percent in the estimated likelihood of failure proneness.

The data collected in project A allowed us to compute two additional measures of congruence, IRC and MR communication congruence. Table 3 reports the odds ratios from the models that included all four measures of congruence. Model I represents the baseline model with only the control factors. Model II incorporates the congruence measures. Overall, the results from model II are consistent with those reported in Table 3. 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. We also replicated the analyses using the data from the entire set of MRs from project A (8,257 MRs). The results were consistent with those reported in Table 3 (model II). However, we did not have IRC-based coordination data for the entire set of MRs. Consequently, we decided to report the analyses based on the subset of MRs that allows us to examine the impact of all four measures of congruence.

TABLE 3
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		0.281*
Geographical Congruence		0.317
MR Congruence		0.209**
IRC Congruence		0.271**
Model Fit		
N	3980	3980
Model $\chi2$	1663**	1859**
df	5	9
Deviance Explained	0.302	0.335
Model Comparison $\chi2$	--	196.24**

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

Summarizing, the results of our analyses suggest that when the relevant work dependencies are adequately managed by the developers, the likelihood of software failures decreases. In other words, higher levels of socio-technical congruence are associated with better software quality (RQ1). Furthermore, the various measures of congruence demonstrate the complementary benefits of different approaches of satisfying coordination needs. Finally, the results also demonstrate that socio-technical congruence is important and valuable in both high maturity and low maturity settings, although the benefits of congruence appear to be greater when maturity is low (RQ2).

7 Socio-Technical Congruence and Software Development Productivity

In this section, we present our empirical examination of the relationship of socio-technical congruence and development productivity across our two projects (RQ3 and RQ4). The contents are organized as follows: We first describe the various measures used in our analyses, followed by a description of the statistical model. We conclude with a presentation of the results.

7.1 Description of the Measures

Past research has identified a number of factors that impact development productivity (e.g., [5], [50], [9]). Some of those are related to characteristics of the 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.

7.1.1 Measuring Development Productivity

Our measure of development productivity is *Resolution Time*, which is defined as the time, in days, it took to resolve a particular MR. We recognize that some modification requests may have longer resolution times for several reasons. For instance, people are working on multiple MRs simultaneously, or an MR was temporarily suspended to address other higher priority work. We addressed these concerns in two different ways. First, we collected several control variables that impact resolution time and they are described later in this section. Second, our measure of productivity accounts only for the time that the MR was assigned to a particular developer. We were able to accurately determine such time periods because the both companies had a process in which modification requests were assigned to developers only when they were actively working on them. Otherwise, in the case of Project A, MRs would be assigned to a generic team identifier. Inspection of a random sample of modification requests suggested that this process followed in the vast majority of cases. In project B, the MR report had an explicit set of fields in which the engineers recorded the times in which they worked on the development tasks. This information enabled us to compute an accurate measure of the time an individual spent in a particular development task, which was also used by the company as part of their resource management systems.

7.1.2 Congruence Measures

The congruence measures were constructed based on the same formulation we proposed in our past work [28], [7]. For each modification request in both projects, we obtained a structural and geographical congruence measure. In the case of project A, we also computed MR and IRC congruence measures.

7.1.3 Additional Control Measures

Past research has reported several additional factors that impact resolution time of development tasks (e.g., [50], [9]). 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 (e.g., [50]) has used lines of code changed as a measure of the size of the modification. However, a comparative analysis of both measures showed the measure using number of files had better explanatory power in the statistical models. 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.

Two measures were constructed to capture attributes of the teams involved in each modification request. *Team load* is a measure of the average workload 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 [57], [58], [5]. Furthermore, experience with the domain area and with the application being developed help accelerate development time [5]. We used data from the software repositories to assess individual-level experience based on the measures proposed by Boh et al. [59]. 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. Finally, the variable *time* captures the month within the project in which the modification request was resolved.

7.2 Description of the Statistical Model

Past research has found that linear and hierarchical linear regression models (e.g., [50], [6]) 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 the resolution time of development tasks using the following linear regression model:

$$\mathrm{Res}olutionTime = \sum_i \beta_i * CongruenceMeasure_i + \sum_j \delta_j * ControlVariable_j + \varepsilon.$$

We report several measures of goodness-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 descriptive statistics and Q-Q plot indicated that several of the variables were highly skewed to the left. The log transformation provided the best approximation to a normal distribution. An analysis of the pairwise correlations among the independent variables as well as a variance inflation analysis suggested no relevant collinearity problems. Finally, we report for each linear regression model the standardized coefficients, also known as beta coefficients. Standardized beta coefficients are measured in standard deviations, instead of the units of the variables, allowing them to be compared to one another. In fact, some researchers use them to compare the relative strength of the various predictors within the model [60]. However, other researchers have called for caution in using betas in such comparisons because a change in standard deviation units might not be equivalent across independent variables [61].

7.3 Results

We performed several linear regression analyses to assess the effect of the congruence measures on resolution time. Table 4 reports the standardized coefficients from the various linear regression models. The results associated with project A have been previously published in our original work [28]. We constructed a baseline model for each project (model I and III) considering only the control factors. The results are consistent with past empirical work in software engineering. Factors such as the size of the

TABLE 4
The Impact of Congruence on the Resolution Time of Modification Requests

	Project A		Project B	
	Model I	**Model II**	**Model III**	**Model IV**
Time	-0.005*	-0.008*	-0.003*	-0.004*
Change Size (log)	0.025**	0.018**	0.014**	0.014**
Team Load	0.033	0.034	-0.009	-0.009
Multiple Locations	0.169**	0.171**	0.084**	0.082*
Shared Work Experience (log)	-0.435**	-0.478**	-0.341**	-0.338**
Component Experience (log)	-0.108**	-0.107**	-0.188**	-0.194**
Structural Congruence		-0.041*		-0.059*
Geographical Congruence		-0.069*		-0.071*
Model Fit				
N	2375	2375	2480	2480
R^2	0.657	0.698	0.463	0.528
Adjusted R^2	0.656	0.697	0.462	0.527
F-test	885.3**	786.3**	418.2**	384.1**

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

modification, familiarity with the software components, familiarity working together with other developers, and geographic distribution are significant factors impacting the resolution time of modification requests in expected directions [59], [50], [9].

We then included the measures of structural and geographical congruence. The results are reported in models II and IV for projects A and B, respectively. Both measures of congruence, structural and geographical, have statistically significant effects. The negative values of their estimated standardized coefficients suggest that higher levels of congruence are associated with a reduction in time to resolve a MR. 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, development productivity increases. Geographical congruence had a positive effect on resolution time, consistent with past research finding distance has detrimental effects on communication (see [9] and [32] for reviews).[5]

The inclusion of the congruence measures in models II and IV results in an improvement in the adjusted R^2 of 6 and 14 percent for projects A and B, respectively. It is important to highlight that the primary purpose of our analyses is to determine the existence of a reliable relationship between congruence and development productivity. Higher improvements in R^2 would be necessary if the focus of the investigation is purely a predictive exercise.

The data collected in project A allowed us to compute two additional measures of congruence: IRC and MR communication congruence. Table 5 reports the results from the models that included all four measures of congruence. The results are consistent with those reported in Table 4. In addition, congruence based on the coordination activities among engineers performed through the MR reports as well as IRC were also statistically significant. These results suggest the usefulness of these tools as additional coordination capabilities that facilitate interdependent work, particularly in a geographically distributed context.

The standardized coefficients reported in Tables 4 and 5 suggest that the impact on the resolution time of modification requests of each individual congruence measure is relatively small, in particular when comparing them to the standardized coefficients associated with experience and geographic distribution. However, the impact of each type of congruence is complementary. Therefore, considering the combined impact of all four measures, we find that congruence is the second most important factor leading to reductions in resolution time of MRs in project A and the third most important in project B, behind the role of the various types of experience.

Summarizing, our results suggest that when the relevant work dependencies are adequately managed by the developers, the resolution time of development tasks decreases. In other words, higher levels of socio-technical congruence are associated with improvements in development productivity (RQ3). In addition, the results also demonstrate that socio-technical congruence is important and valuable not just in a novel and less mature development setting such as project A but also in a significantly more mature development setting such as project B (RQ4).

8 A Closer Look at the Temporal Evolution of Coordination Requirements

In Sections 6 and 7, we examined our research questions and our analyses showed that matching the work coordination needs that emerge from the technical dependencies with appropriate coordinative actions helps improve software quality and development productivity. Furthermore, those results are consistent across two projects with very distinct characteristics in terms of product and process maturity. In this section, we take a closer look at how the coordination needs of the development organization evolve over time. This is of particular interest because highly

5. The measures of structural and geographical congruence could be affected by personnel turnover and mobility across teams. In project A, archival data showed a yearly turnover rate of only 3 percent and an intergroup mobility rate of less than 1 percent. Removing the MRs that involved individuals who left the company or changed group membership yielded consistent results to those reported in Tables 4 and 5. Unfortunately, we did not have access to the necessary data in project B but based on interviews with project members, turnover was perceived to be negligible.

TABLE 5
The Impact of Congruence on Resolution Time When Considering Additional Coordination Capabilities in Project A

	Model I	Model II
Time	-0.005*	-0.006*
Change Size (log)	0.025**	0.018**
Team Load	0.033	0.031
Multiple Locations	0.169**	0.171**
Shared Work Experience (log)	-0.435**	-0.436**
Component Experience (log)	-0.108**	-0.108**
Structural Congruence		-0.041*
Geographical Congruence		-0.070*
MR Congruence		-0.034*
IRC Congruence		-0.043*
Model Fit		
N	2375	2375
R^2	0.657	0.748
Adjusted R^2	0.656	0.747
F-test	885.3**	763.1**

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

volatile coordination needs tend to render traditional coordination approaches such as standardized processes inadequate [47], [12], [19], [29]. Past research has examined the evolution of coordination needs over the life cycle of a development project. Our past work [7] showed that coordination needs changed substantially on a weekly basis and that a sizeable portion of those coordination needs tended to involve different formal teams. Costa et al. [62] extended our original analyses [7] to examine the scale and range of coordination needs across organizational boundaries such as formal teams, development locations, and time zones.

In this paper, we extend those two previous works in the following way: First, we extended the Costa et al. [62] analyses by examining the evolution and the oscillation of new coordination need on a monthly basis. These analyses allow us to understand the extent to which new coordination needs could represent a major challenge of software projects.[6] Second, we replicated the evolution analysis across different groups of developers reported in our earlier work [7].[7]

Fig. 2 depicts the evolution of the proportion of new coordination needs that, on average, each engineer faces 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 proportion of new coordination needs is represented by the bars in darker color. We identified new coordination needs by comparing the coordination requirements matrices corresponding to month t against month $t-1$. Then, the cells in the C_R matrix from month t that are nonzero and have a corresponding zero value in the C_R matrix from month $t-1$ constitute new coordination requirements at time t that did not exist at time $t-1$. Fig. 2 also shows the proportion of new coordination needs that involve individuals in other formal organizational groups (bars in light color in Fig. 2). In our research settings, that proportion implies that the coordination requirement involves two distinct architectural components. When all the new coordination needs in a particular month are with individuals outside the person's formal team, the two bars representing the two measures overlap. Hence, Fig. 2 only shows the bar in light color corresponding to the average proportion of new coordination needs that involve individuals in other formal organizational group.

On average and over the life cycle of the project, each engineer in project A faced 2.9 new coordination requirements out of 27.2 total coordination needs on a monthly basis. In project B, each engineer faced 5.3 new coordination requirements out of 25.8 total coordination needs on a monthly basis. Although, the overall averages seem somewhat small, it is clear from Fig. 2 that the average amount of new coordination requirements faced by a developer constitute a substantial proportion of all coordination needs at several points in time in the projects. Furthermore, those new coordination needs tend to be quite volatile over time in both projects, suggesting that mature development settings such as project B could be at a disadvantage. The same volatile patterns are exhibited by those new coordination needs that involve different organizational teams. Consequently, those coordination needs are associated with development work involving different architectural components, which arguably introduce another dimension of complexity in the context of coordinating development work.

In addition to highly volatile patterns, Fig. 2 highlights points in the life cycle of a project where new coordination needs significantly exceed average levels. For example, in project A we observe that the proportion of new coordination needs is above 30 percent on months 24, 27, and 28, while in project B, months 51 and 53 exceed those levels as well. We took a closer look at the characteristics of the development tasks to determine the source of such patterns. We examined the descriptions and comments stored in the MR reports. We also examined the attributes of the commits in the version control system such as the number of files, whether files were being removed, moved, or renamed, as well as whether significant portions of the

6. Projects A and B in this paper correspond to projects A and B in [22]. Thompson [22] reported average levels of volatility and oscillation whereas in this paper we focused on monthly level data and how it evolved over time.

7. Galibrath [19] reported similar results as those in Figs. 4 and 5 but using a subset of the data (only 809 MRs) used in this study.

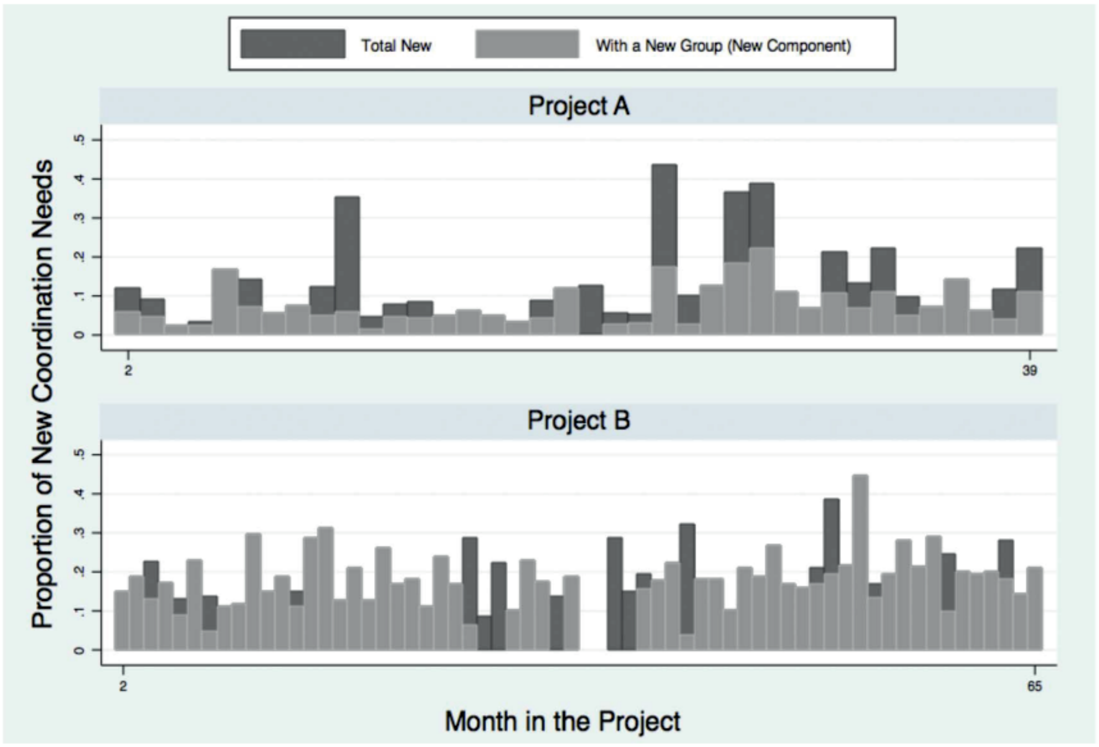

Fig. 2. The evolution of new coordination requirements on a monthly basis.

code were being moved across files. The goal of this analysis was to understand if those above average levels in new coordination need were a byproduct of cross-cutting features being implemented, decisions to rearchitect, or refactor parts of the software systems. In project A, we found that the development tasks within three months of the points of interest tended, on average, to involve higher numbers of architectural components (1.78 versus 3.1). However, the tasks did not involve the development of obvious "cross-cutting" concerns. The tasks represented core features of the product that impacted several parts of the software system. In project B, we did not find any particular characteristics of the tasks that would explain the above average levels of coordination needs.

We perform additional analyses to gain further insight on the challenges associated with the volatility of new coordination requirements. Specifically, we examined the degree to which the new coordination needs oscillate among small groups of people and over short periods of time or whether they exhibit more complex patterns. When coordination needs oscillate among a small set of individuals over a short period of time (e.g., every other month), we could argue that the volatility of the coordination needs might not be a major challenge. The reason is that individuals are likely develop implicit coordination mechanisms to handle those dependencies [50]. On the other hand, when new coordination needs involve constantly new actors, the coordination effort increases and the likelihood for misunderstanding, errors, and lower productivity increases. In particular, we are interested in those new coordination needs involving at least two distinct architectural components. In other words, those coordination needs involving other formal teams.

Fig. 3 depicts the proportions of the new out-of-group coordination requirements computed not just by comparing C_R matrices from time t and $t-1$ but also comparing month t against the coordination needs from months $t-2$ and $t-3$. A new coordination need with respect to $t-i$ means an engineer has not had coordination needs with a particular person in the last i months. The three different measures are depicted in a form of a bar, with the darker shade corresponding to the $t-1$ new out-of-group coordination needs while the lightest shade corresponds to the $t-3$ measure. This graphical approach allows us to clearly observe the proportions of out-of-group coordination needs that are new relative to the past month, the past two months, and the past three months. On average over the lifetime of a project, 52.6 percent of the new out-of-group coordination needs in project A are among individuals that have not been interdependent in the last month (Fig. 3, "relative to month $t-1$" data series). In project B, that percentage of new coordination needs is 56.7 percent. More importantly, 29.1 and 18.2 percent of the new out-of-group coordination requirements in project A are among developers that have not had coordination needs in the last two and three months, respectively. In project B, those percentages are 28.2 and 15.1 percent, respectively. Fig. 3 also shows many instances where the proportion of this challenging set of coordination needs represent more than 50 percent of the new coordination needs faced by an individual.

Summarizing, our results show that new coordination needs tend to be quite volatile over the life of both projects despite their differences in terms of product and process maturity. More importantly, coordination needs that cross formal organizational boundaries (e.g., formal team) are also quite common. Furthermore, in many instances they involve engineers that have not been interdependent for at

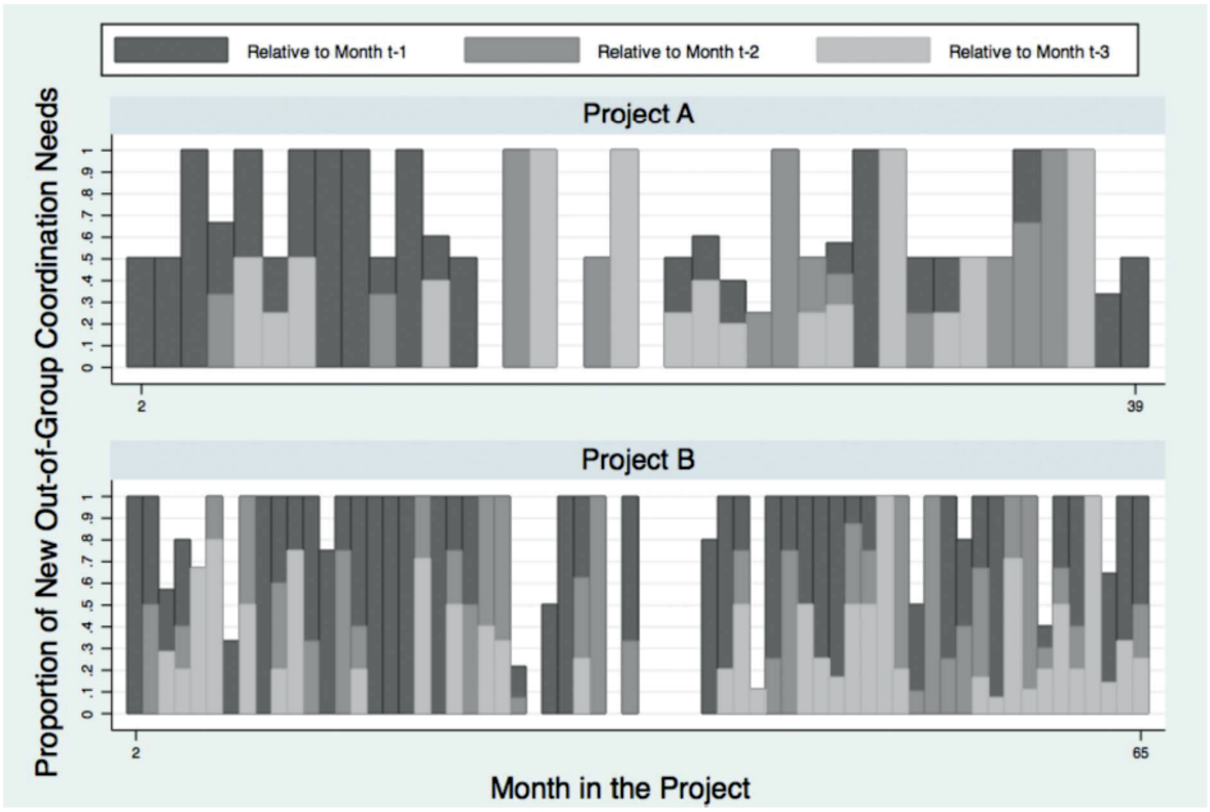

Fig. 3. The "Newness" of new out-of-group coordination needs on a monthly basis.

least three months. Combined, these results suggest the need for new coordination mechanisms that complement traditional coordination methods (e.g., standardized processes) in assisting project members in navigating the complex landscape of work dependencies.

9 A Closer Look at the Temporal Evolution of Congruence

In earlier sections, we showed that the various measures of congruence are associated with improvements in development productivity and software quality. The development productivity analysis showed evidence of learning effects. In fact, it is well established that individuals, groups, and organizations are able to learn over time to perform their task better [63]. In our context, as developers gain experience in different aspects of the development project, we could expect that developers, over time, are better able to identify and manage emerging work dependencies. Examples of the experience gained include understanding how the various components of the system being developed relate to each, learning who is working in the various parts of a system, and learning how to work with each other. The benefits accrued from learning, however, 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). In this section, we take a closer look at the temporal evolution of our socio-technical congruence measures.

In order to examine the evolutionary patterns of congruence, we constructed measures of congruence on a semi-annual basis, which resulted in six semesters for project A and 10 semesters for project B. We considered all the modification requests started and completed within each semester of each project covered by our data. Combining that data with the logical dependencies data corresponding to the same semester, we constructed a semi-annual C_R matrix, as discussed in Section 5. Fig. 4 shows the average level of the structural and geographical congruence over time in projects A and B. We observe that structural congruence (solid line) declines significantly over time in both projects. This decline could be interpreted as a deterioration of the homomorphic relationship between product and work structures posited by the modularity theoretical argument [15], [18]. In other words, the benefits of modularizing a software system to reduce the interdependency among development teams seem to diminish over time. Regarding geographical congruence, Fig. 4 shows that it remains relatively stable in both projects (dashed line) suggesting that the coordination needs across locations did not change significantly 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.

Our previous work [7] showed that the evolutionary patterns of the congruence differ significantly between those developers that contribute larger portions to the development effort and the rest of the developers. We replicated their analysis and Fig. 5 depicts the results. In project A, we found that 50 percent of the modifications made to the system were done by only 18 (15 percent) developers, while in project B, 107 developers (28 percent) contributed 50 percent of the modifications. We labeled those developers as "top-contributors" and we labeled the rest of the developers as "other-developers." In project A, the top contributors were involved in tasks that required

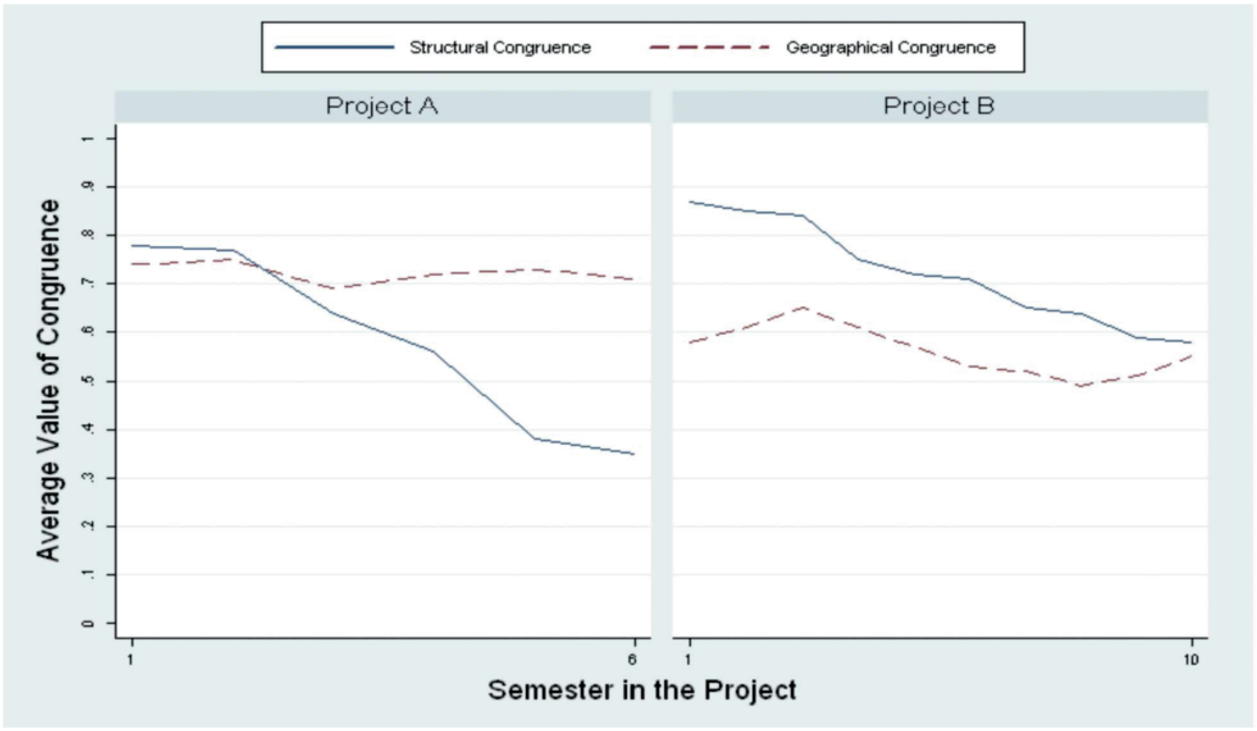

Fig. 4. The evolution of structural and geographical congruence.

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 other developers seem not to use 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-based measures. It is important to point out that the usage of the IRC and MR tracking tools in project A was pervasive and interviews with 12 engineers suggested that they were very important mechanisms for information exchange and coordination related to development tasks. In project B, we observe

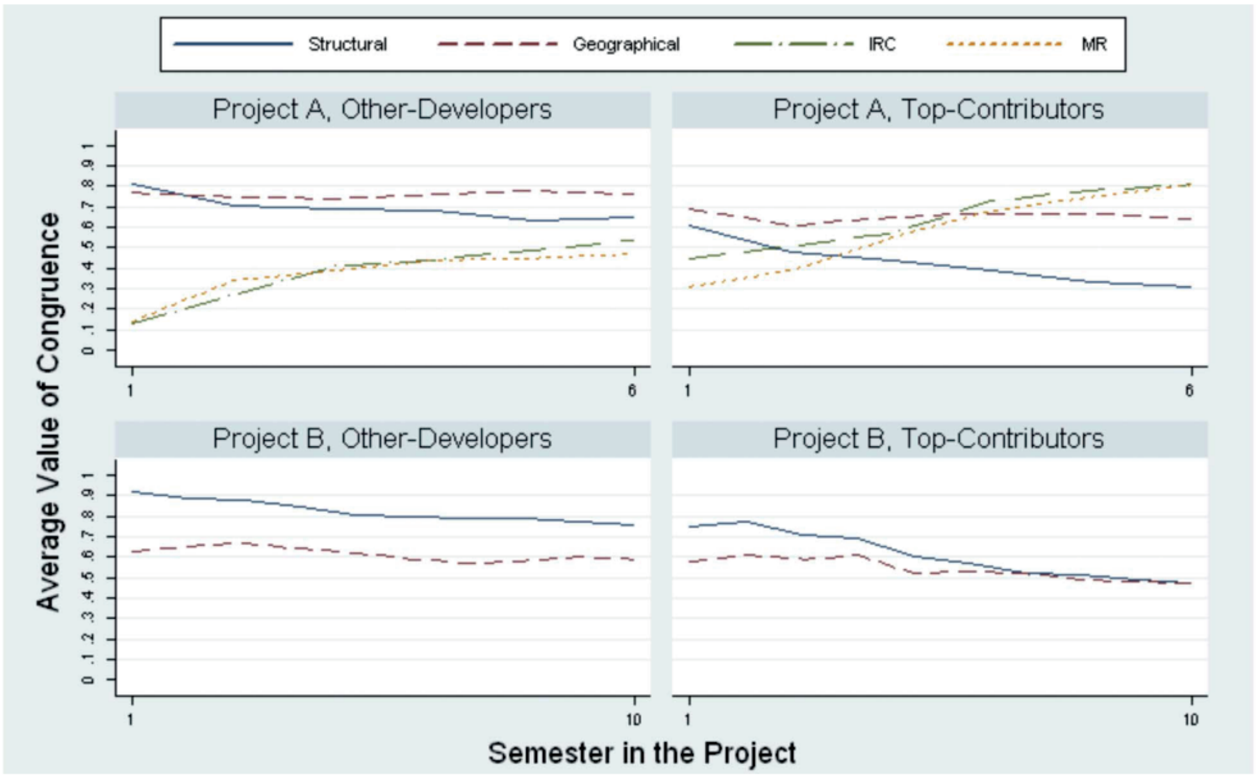

Fig. 5. The evolution of congruence for top-performing and other developers.

similar patterns where top contributors tend to be involved in tasks that require more cross-team coordination, particularly as the project evolves.

Summarizing, the patterns shown in Figs. 4 and 5 represent two additional and important results. First, the decreasing levels of structural congruence suggest that the benefits of modularization diminish as a project matures. Second, top contributors tend to exhibit very different coordinative actions relative to other developers, suggesting that the traditional perspective in software engineering relating only cognitive ability and experience to contributions [58], [5] may not capture all of the important attributes of top contributors since their performance seems to have a substantial social component as well.

10 Discussion

This paper studied the relationship between socio-technical congruence and critical software development outcomes such as software quality and development productivity. We addressed four specific research questions. First, we examined the impact of congruence on software quality (RQ 1). 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 software failures. Second, we also studied the relationship between congruence and development productivity (RQ 3). Our analyses showed that when engineers identified and managed the *relevant* coordination needs, development productivity improved. Finally, our analyses were performed on two different large-scale projects from two different companies with very distinct characteristics in terms of product and process maturity (RQs 2 and 4). Our results showed that considering dependencies and adequately managing them is critical in mature development settings as well as less mature contexts.

Our descriptive analyses of the temporal evolution of coordination needs and congruence provided additional important results. We showed that new coordination needs are quite volatile and they also tend to cross formal teams. We observed the same results across our two projects. These results suggest that mature development settings also face significant coordination challenges despite the lower levels of uncertainty associated with their products and more mature development processes. Combined, these findings demonstrate the dynamic nature of work dependencies that exist in software development projects.

Second, our measure of structural congruence 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., [15]). As demonstrated by 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 a group of developers in project A tended to coordinate more congruently over communication mediums such as IRC and the MR tracking system. Those patterns occurred as the coordination mechanism provided by the formal organization increasingly failed to match the developer's coordination needs.

10.1 Limitations

It is also important to highlight some of the limitations of the work reported in this paper. First, the congruence measures are contingent on assumptions about the software development processes used in the organization as well as usage patterns of tools that assist the development effort such as defect tracking and version control systems. One key assumption is the possibility of identifying 1) the set of source code files that were changed as part of a modification request and 2) the developers who made those changes. Both our research settings, as described earlier, had a collection of tools that allow a very detailed tracking and linkage of changes in the version control system with task tracking systems. However, we recognized that such approaches might not be commonly used in industry or open source projects. Furthermore, particular processes (e.g., file ownership policy) may also impact the measurement of congruence. However, our projects did not exhibit such a situation.

A second limitation of our work is that we did consider all forms of coordination, such as telephone, e-mail communication, and certain forms of documentation. Unfortunately, we were not able to collect such data in both of our research settings. It is worth pointing out that in project A, interviews with 17 project members revealed that the project used face-to-face meetings and conference calls for architectural and design meetings. However, the comments in the MR tracking system and IRC were the primary means of communication for task-related discussion as well as exchanging technical information. Therefore, we think that our measures of MR and IRC congruence capture a very relevant aspect of the coordination activities in project A.

10.2 Implications for Future Research

The results reported in this paper provide a number of directions for future research activities. First, our results have important implications for tools to support development organizations manage the dynamic and evolving web of work dependencies in software development projects. First, collaboration and awareness tools such as Palantir [67], Ariadne [68], FastDASH [69], and TeamRooms [70] could use our coordination requirements needs and congruence measures to provide complementary dependency information to such tools. Recent tools such as Tesseract [71] and Ensemble [72] have implemented various aspects of the congruence approach. However, they have not yet been empirically evaluated in the context of coordination and awareness. A second type of tools that could benefit from the work presented in this paper builds design principles from social computing and expertise finder tools. For example, Codebook [73] presents an approach to mine software repositories such that users could search the graph of interrelated individuals and artifacts. Our results suggest that tools such as Codebook could be improved by using the measures of coordination needs and congruence to recommend project members or artifacts that are highly interrelated to a person's current tasks. Finally, separating the coordination needs across organizational or geographical boundaries can be used to enhance the capabilities of collaboration tools for large-scale projects. In sum, our congruence approach allows tools to identify the relevant dependencies and raise awareness among the project members about those dependencies. Then, tools would be in a better position to manage the issues of scale associated with identification, management, and presentation of dependency information.

Second, the work reported in this paper has focused on development tasks. Certainly software projects involve other types of tasks and stakeholders. Recent research has shown the value of socio-technical congruence in other contexts. Wagstrom et al. [75] showed that congruence is associated with higher individual-level performance in open source projects. Damian et al. [76] showed the applicability of the socio-technical congruence approach in requirement engineering tasks. Future research should consider the use of the congruence approach across other types of tasks that take place in a software projects as well as integrating other stakeholders (e.g., architects, designers, or testers) and artifacts. In a related vein, the general nature of the congruence approach opens the door for the analyses of numerous other forms of technical dependencies. For example, dependencies could be grouped around product features or around "cross-cutting" concerns [23], which might not necessarily match the source code file or component-based representations of technical dependencies, and the implications on coordination of those units could then be examined.

Finally, the nature of the coordinative actions may vary across pairs of individuals, groups, or when involving different parts of a software systems. For example, coordination activities might take place among individuals that might not be explicitly interdependent. Recent research has considered variants to the congruence approach in order to gain more insight on the patterns of coordination needs, coordinative actions, and their impact. Ehrlich et al. [77] examined the mismatches between coordination needs and action. The authors found that geographically distributed pairs of individuals tended to have higher numbers of unsatisfied coordination needs. Kwan et al.[78] proposed a weighted measure of congruence to gain a better understanding of the relative impact of the strength of the coordination needs and the corresponding coordination activities. Wagstrom et al. [75] proposed and evaluated an individual-level measure of congruence that allows the examination of the relationship between a person's congruence patterns and his/her individual-level performance. Furthermore, the authors found that communication among individuals who did not have coordination needs among them did not impact the resolution time of defects. However, Sosa et al. [38] in an analysis of a complex product development project found that noncongruent communication is likely when indirect technical dependencies exist and they tend to be valuable from a quality point of view. Future research should examine in more detail the role noncongruent communication as well as considering further empirical evaluations of the various proposed extensions to our congruence metric.

REFERENCES

[1] C. Ebert and C. Jones, "Embedded Software: Facts, Figures, and Future," *Computer*, vol. 42, no. 4, pp. 42-52, Apr. 2009.

[2] J.D. Herbsleb and D. Moitra, "Guest Editors' Introduction: Global Software Development," *IEEE Software*, vol. 18, no. 2, pp. 16-20, Mar./Apr. 2001.

[3] D.W. Karolak, *Global Software Development: Managing Virtual Teams and Environments*. IEEE CS Press, 1998.

[4] L. Northrop, P.H. Feiler, B. Pollak, and D. Pipitone, *Ultra-Large-Scale Systems: The Software Challenge of the Future*. Software Eng. Inst., Carnegie Mellon Univ., 2006.

[5] B. Curtis, H. Krasner, and N. Iscoe, "A Field Study of the Software Design Process for Large Systems," *Comm. ACM*, vol. 31, pp. 1268-1287, 1988.

[6] R.E. Kraut and L.A. Streeter, "Coordination in Software Development," *Comm. ACM*, vol. 38, pp. 69-81, 1995.

[7] M. Cataldo, P.A. Wagstrom, J.D. Herbsleb, and K.M. Carley, "Identification of Coordination Requirements: Implications for the Design of Collaboration and Awareness Tools," *Proc. 20th Anniversary Conf. Computer Supported Cooperative Work*, 2006.

[8] R.E. Grinter, J.D. Herbsleb, and D.E. Perry, "The Geography of Coordination: Dealing with Distance in R&D Work," *Proc. Int'l ACM SIGGROUP Conf. Supporting Group Work*, 1999.

[9] J.D. Herbsleb and A. Mockus, "An Empirical Study of Speed and Communication in Globally Distributed Software Development," *IEEE Trans. Software Eng.*, vol. 29, no. 6, pp. 481-494, June 2003.

[10] C.R.B.D. Souza, D. Redmiles, L.-T. Cheng, D. Millen, and J. Patterson, "How a Good Software Practice Thwarts Collaboration: The Multiple Roles of APIs in Software Development," *Proc. 12th ACM SIGSOFT Int'l Symp. Foundations of Software Eng.*, 2004.

[11] T.R. Browning and R.V. Ramasesh, "A Survey of Activity Network-Based Process Models for Managing Product Development Projects," *Production and Operations Management*, vol. 16, pp. 217-240, 2007.

[12] S. Faraj and Y. Xiao, "Coordination in Fast-Response Organizations," *Management Science*, vol. 52, pp. 1155-1169, 2006.

[13] H.A. Simon, "The Architecture of Complexity," *Proc. Am. Philosophical Soc.*, vol. 106, pp. 467-482, 1962.

[14] K.T. Ulrich and S. Eppinger, *Product Design and Development*, fourth ed. McGraw-Hill, 2007.

[15] C.Y. Baldwin and K.B. Clark, *Design Rules*. MIT Press, 2000.

[16] S.D. Eppinger, D.E. Whitney, R.P. Smith, and D.A. Gebala, "A Model-Based Method for Organizing Tasks in Product Development," *Research in Eng. Design*, vol. 6, pp. 1-13, 1994.

[17] K.J. Sullivan, W.G. Griswold, Y. Cai, and B. Hallen, "The Structure and Value of Modularity in Software Design," *Proc. Eighth European Software Eng. Conf. Held Jointly with Ninth ACM SIGSOFT Int'l Symp. Foundations of Software Eng.*, 2001.

[18] D.L. Parnas, "On the Criteria to Be Used in Decomposing Systems into Modules," *Comm. ACM*, vol. 15, pp. 1053-1058, 1972.

[19] J. Galbraith, *Designing Complex Organizations*. Addison-Wesley Publishing Co., 1973.

[20] T.W. Malone and K. Crowston, "The Interdisciplinary Study of Coordination," *ACM Computing Surveys*, vol. 26, pp. 87-119, 1994.

[21] J.G. March and H.A. Simon, *Organizations*. Wiley, 1958.

[22] J.D. Thompson, *Organizations in Action; Social Science Bases of Administrative Theory*. McGraw-Hill, 1967.

[23] G. Kiczales and M. Mezini, "Aspect-Oriented Programming and Modular Reasoning," *Proc. 27th Int'l Conf. Software Eng.*, 2005.

[24] J.H. Gittell, "Coordinating Mechanisms in Care Provider Groups: Relational Coordination as a Mediator and Input Uncertainty as a Moderator of Performance Effects," *Management Science*, vol. 48, pp. 1408-1426, 2002.

[25] M.E. Conway, "How Do Committees Invent?" *Datamation*, vol. 14, pp. 28-31, 1968.

[26] M. Cataldo, M. Bass, J.D. Herbsleb, and L. Bass, "On Coordination Mechanisms in Global Software Development," *Proc. Int'l Conf. Global Software Eng.*, 2007.

[27] C.R.B.D. Souza and D.F. Redmiles, "An Empirical Study of Software Developers' Management of Dependencies and Changes," *Proc. 30th Int'l Conf. Software Eng.*, 2008.

[28] M. Cataldo, J.D. Herbsleb, and K.M. Carley, "Socio-Technical Congruence: A Framework for Assessing the Impact of Technical and Work Dependencies on Software Development Productivity," *Proc. ACM/IEEE Second Int'l Symp. Empirical Software Eng. and Measurement*, 2008.

[29] N. Staudenmayer, *Managing Multiple Interdependencies in Large Scale Software Development Projects.* Sloan School of Management, Massachusetts Inst. of Technology, 1997.

[30] W.R. Scott, *Organizations: Rational, Natural, and Open Systems,* fourth ed. Prentice Hall, 1998.

[31] M. Cataldo and S. Nambiar, "On the Relationship between Process Maturity and Geographic Distribution: An Empirical Analysis of Their Impact on Software Quality," *Proc. Seventh Joint Meeting of the European Software Eng. Conf. and the ACM SIGSOFT Symp. Foundations of Software Eng.*, 2009.

[32] G.M. Olson and J.S. Olson, "Distance Matters," *Human, & Computer Interaction,* vol. 15, pp. 139-178, 2000.

[33] R. Garud, A. Kumaraswamy, and R.N. Langlois, *Managing in the Modular Age: Architectures, Networks, and Organizations.* Blackwell, 2003.

[34] E. von Hippel, "Task Partitioning: An Innovation Process Variable," *Research Policy,* vol. 19, pp. 407-418, 1990.

[35] T.R. Browning, "Applying the Design Structure Matrix to System Decomposition and Integration Problems: A Review and New Directions," *IEEE Trans. Eng. Management*, vol. 48, no. 3, pp. 292-306, Aug. 2001.

[36] M. Shaw and D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline.* Prentice Hall, 1996.

[37] A. MacCormack and R. Verganti, "Managing the Sources of Uncertainty: Matching Process and Context in Software Development," *J. Product Innovation Management*, vol. 20, pp. 217-232, 2003.

[38] M.E. Sosa, S.D. Eppinger, and C.M. Rowles, "The Misalignment of Product Architecture and Organizational Structure in Complex Product Development," *Management Science,* vol. 50, pp. 1674-1689, 2004.

[39] B. Gokpinar, W.J. Hopp, and S.M.R. Iravani, "The Impact of Misalignment of Organizational Structure and Product Architecture on Quality in Complex Product Development," *Management Science,* vol. 56, pp. 468-484, 2010.

[40] J.D. Herbsleb, A. Mockus, and J.A. Roberts, "Collaboration in Software Engineering Projects: A Theory of Coordination," *Proc. Int'l Conf. Information Systems,* 2006.

[41] M. Cataldo, A. Mockus, J.A. Roberts, and J.D. Herbsleb, "Software Dependencies, Work Dependencies, and Their Impact on Failures," *IEEE Trans. Software Eng.,* vol. 35, no. 6, pp. 864-878, Nov./Dec. 2009.

[42] B.W. Boehm and R. Turner, *Balancing Agility and Discipline: A Guide for the Perplexed.* Addison-Wesley, 2004.

[43] D.E. Perry, N.A. Staudenmayer, and L.G. Votta, "People, Organizations, and Process Improvement," *IEEE Software,* vol. 11, no. 4, pp. 36-45, July 1994.

[44] S.G. Eick, T.L. Graves, A.F. Karr, J.S. Marron, and A. Mockus, "Does Code Decay? Assessing the Evidence from Change Management Data," *IEEE Trans. Software Eng.,* vol. 27, no. 1, pp. 1-12, Jan. 2001.

[45] A. Fuggetta, "Software Process: A Roadmap," *Proc. Conf. Future of Software Eng.,* 2000.

[46] D.E. Harter, M.S. Krishnan, and S.A. Slaughter, "Effects of Process Maturity on Quality, Cycle Time, and Effort in Software Product Development," *Management Science,* vol. 46, pp. 451-466, 2000.

[47] R.L. Daft and R.H. Lengel, "Organizational Information Requirements, Media Richness and Structural Design," *Management Science,* vol. 32, pp. 554-571, 1986.

[48] R.M. Burton, B. Obel, S. Hunter, M. Søndergaard, and D. Døjbak, *Strategic Organizational Diagnosis and Design: Developing Theory for Application,* second ed. Kluwer Academic Publishers, 1998.

[49] K.M. Carley and Y. Ren, "Tradeoffs between Performance and Adaptability for C3I Architectures," *Proc. Sixth Int'l Command and Control Research and Technology Symp.,* 2001.

[50] J.A. Espinosa, S.A. Slaughter, R.E. Kraut, and J.D. Herbsleb, "Familiarity, Complexity, and Team Performance in Geographically Distributed Software Development," *Organization Science,* vol. 18, pp. 613-630, 2007.

[51] T.J. Allen, *Managing the Flow of Technology: Technology Transfer and the Dissemination of Technological Information within the R&D Organization.* MIT Press, 1977.

[52] N. Nagappan and T. Ball, "Using Software Dependencies and Churn Metrics to Predict Field Failures: An Empirical Case Study," *Proc. First Int'l Symp. Empirical Software Eng. and Measurement,* 2007.

[53] N. Nagappan, B. Murphy, and V. Basili, "The Influence of Organizational Structure on Software Quality: An Empirical Case Study," *Proc. 30th Int'l Conf. Software Eng.,* 2008.

[54] L.C. Briand, J. Wüst, J.W. Daly, and D.V. Porter, "Exploring the Relationship between Design Measures and Software Quality in Object-Oriented Systems," *J. Systems Software,* vol. 51, pp. 245-273, 2000.

[55] T.L. Graves, A.F. Karr, J.S. Marron, and H. Siy, "Predicting Fault Incidence Using Software Change History," *IEEE Trans. Software Eng.,* vol. 26, no. 7, pp. 653-661, July 2000.

[56] D.J. Watts, *Small Worlds: The Dynamics of Networks between Order and Randomness.* Princeton Univ. Press, 1999.

[57] F.P. Brooks, *The Mythical Man-Month: Essays on Software Engineering,* anniversary ed. Addison-Wesley Pub. Co., 1995.

[58] B. Curtis, *Human Factors in Software Development.* IEEE CS Press, 1981.

[59] W.F. Boh, S.A. Slaughter, and J.A. Espinosa, "Learning from Experience in Software Development: A Multilevel Analysis," *Management Science,* vol. 53, pp. 1315-1331, 2007.

[60] J. Cohen and P. Cohen, *Applied Multiple Regression/Correlation Analysis for the Behavioral Sciences.* Lawrence Elrbaum, 1983.

[61] L. Schroeder, D.L. Sjoquist, and P.E. Stephan, *Understanding Regression Analysis: An Introductory Guide.* Sage Publications, 1986.

[62] J.M. Costa, M. Cataldo, and C.R.B. de Souza, "The Scale and Evolution of Coordination Needs in Large-Scale Distributed Projects: Implications for the Future Generation of Collaborative Tools," *Proc. Int'l Conf. Computer-Human Interaction,* 2011.

[63] L. Argote, *Organizational Learning: Creating, Retaining, and Transferring Knowledge.* Kluwer Academic, 1999.

[64] M.E. Sosa, "A Structured Approach to Predicting and Managing Technical Interactions in Software Development," *Research in Eng. Design,* vol. 19, pp. 47-70, 2008.

[65] R.M. Henderson and K.B. Clark, "Architectural Innovation: The Reconfiguration of Existing Product Technologies and the Failure of Established Firms," *Administrative Science Quarterly,* vol. 35, pp. 9-30, 1990.

[66] M. Hoegl, K. Weinkauf, and H.G. Gemuenden, "Interteam Coordination, Project Commitment, and Teamwork in Multiteam R&D Projects: A Longitudinal Study," *Organization Science,* vol. 15, pp. 38-55, 2004.

[67] A. Sarma, Z. Noroozi, and A.V.D. Hoek, "Palantir: Raising Awareness among Configuration Management Workspaces," *Proc. 25th Int'l Conf. Software Eng.,* 2003.

[68] E. Trainer, S. Quirk, C.D. Souza, and D. Redmiles, "Bridging the Gap between Technical and Social Dependencies with Ariadne," *Proc. OOPSLA Workshop Eclipse Technology eXchange,* 2005.

[69] J.T. Biehl, M. Czerwinski, G. Smith, and G.G. Robertson, "FASTDash: A Visual Dashboard for Fostering Awareness in Software Teams," *Proc. SIGCHI Conf. Human Factors in Computing Systems,* 2007.

[70] M. Roseman and S. Greenberg, "TeamRooms: Network Places for Collaboration," *Proc. ACM Conf. Computer Supported Cooperative Work,* 1996.

[71] A. Sarma, L. Maccherone, P. Wagstrom, and J. Herbsleb, "Tesseract: Interactive Visual Exploration of Socio-Technical Relationships in Software Development," *Proc. 31st Int'l Conf. Software Eng.,* 2009.

[72] P.F. Xiang, A.T.T. Ying, P. Cheng, Y.B. Dang, K. Ehrlich, M.E. Helander, P.M. Matchen, A. Empere, P.L. Tarr, C. Williams, and S.X. Yang, "Ensemble: A Recommendation Tool for Promoting Communication in Software Teams," *Proc. Int'l Workshop Recommendation Systems for Software Eng.,* 2008.

[73] A. Begel, Y.P. Khoo, and T. Zimmermann, "Codebook: Discovering and Exploiting Relationships in Software Repositories," *Proc. ACM/IEEE 32nd Int'l Conf. Software Eng.,* vol. 1, 2010.

[74] R. Allen and D. Garlan, "A Formal Basis for Architectural Connection," *ACM Trans. Software Eng. Methodology,* vol. 6, pp. 213-249, 1997.

[75] P. Wagstrom, J.D. Herbsleb, and K.M. Carley, "Communication, Team Perforance and the Individual: Bridging Technical Dependencies," *Proc. Academy of Management Ann. Meeting,* 2010.

[76] D. Damian, I. Kwan, and S. Marczak, "Requirements-Driven Collaboration: Leveraging the Invisible Relationships Between Requirements and People," *Collaborative Software Engineering,* A. Finkelstein, et al., eds., Springer-Verlag, 2010.

[77] K. Ehrlich, M.E. Helander, G. Valetto, S. Davis, and C. Williams, "An Analysis of Congruence Gaps and Their Effect on Distributed Software Development," *Proc. First Int'l Workshop Socio-Technical Congruence*, 2008.

[78] I. Kwan, A. Schroeter, and D. Damian, "A Weighted Congruence Measure," *Proc. Second Int'l Workshop Socio-Technical Congruence*, 2009.

Marcelo Cataldo received the BS degree in information systems from the Universidad Tecnologica Nacional, Argentina, in 1996 and the MS degree in information networking from Carnegie Mellon University in 2000. He also received the MS and PhD degrees in computation, organizations, and society from Carnegie Mellon University in 2007. His research interests include geographically distributed software development with special focus on the relationship between the software architecture and the organizational structure in large-scale software development projects. He is a senior researcher at Robert Bosch's Research and Technology Center.

James D. Herbsleb received the JD degree in 1980 and the PhD degree in psychology in 1984 from the University of Nebraska. He also received the MS degree in computer science, in 1991 from the University of Michigan. He is a professor of computer science and the director of the Software Industry Center at Carnegie Mellon University. His research interests lie primarily in the intersection of software engineering and computer-supported cooperative work, focusing on such areas as geographically distributed development teams, open source software development, and more generally on coordination in software engineering.