A Fine-grain Measure of Coordination: Implications for the Design of Collaboration and Awareness Tools

Marcelo Cataldo[1] Patrick A. Wagstrom[2] James D. Herbsleb[1] Kathleen M. Carley[1]

[1] Institute for Software Research International

[2] Department of Engineering and Public Policy

Carnegie Mellon University

Pittsburgh, PA 15213

{mcataldo,pwagstro}@andrew.cmu.edu jdh@cs.cmu.edu kathleen.carley@cmu.edu

ABSTRACT

In this paper, we develop a technique to measure how congruent the actual organizational communication channels are relative to the coordination requirement imposed by the dependencies among tasks. We examine the role of congruence in the context of a closed source project of a large distributed system. Our results show that congruence helped reduce resolution time of software modification requests. We also explore the evolution of congruence across several releases of the product. As task dependencies changed over time, we found that developers, in particular the most productive ones, change their patterns of usage of communication technologies such as Internet Relay Chat (IRC) and task-tracking systems. Finally, we discuss the practical implications of our technique for the design of collaborative and awareness tools.

1. INTRODUCTION

Coordination arises as a response to interdependent activities. The literature on organizational theory has proposed several stylized coordination mechanisms. For instance, Thompson (1967) argued standardization, plan and mutual adjustment are appropriate coordination mechanisms when dependencies follow pooled, sequential or reciprocal patterns, respectively. Crowston (1991) extended that work and developed a taxonomy of dependency types and coordination mechanisms appropriate for each type. These high-level conceptualizations have been particularly useful in organizational design. However, several types of work, in particular knowledge-intensive activities, are full of fine-grain dependencies that might change on a daily or hourly basis. Designing tools that support rapidly shifting coordination needs require a more specific level of analysis than what the traditional views of coordination provide.

In this paper, we develop a technique to measure the fit between the task dependencies and the communication activities performed by individuals. We refer to such measure as congruence. Using data from a software development project, our analysis found that congruence helps reduce the amount of time required to perform tasks. In addition, the results show that over time individuals learn to use tools in ways that are more congruent with the work they perform. Moreover, the most productive workers reach higher levels of congruence than the less productive ones. These results suggest that congruence can provide a basis for understanding the nature of collaborations that should be supported given a particular set of task dependencies. In addition, congruence could be computed automatically, given a person's activities, to determine the set of people he or she should be aware of or communicate with. Then, collaboration technologies could use that data in a variety of ways to understand what needs to be brought into the user experience.

2. COORDINATION REQUIREMENTS AND CONGRUENCE

In the organizational theory literature, the concept of congruence, or “fit”, is a well developed idea and it typically refers to the match between a particular organizational design and the organization’s ability to carry out a task

(Burton & Obel, 1998). The definition or selection of an organizational structure is influenced, among several factors, by the interdependencies among tasks and, consequently, by the coordination requirements. March and Simon (1958) argued that coordination encompasses more than just a traditional division of labor and assignment approach. They proposed numerous mechanisms such the division of the task in nearly independent parts or schedules and feedback mechanisms when interdependence is unavoidable. Thompson (1967) extended March and Simon's work by matching three mechanisms, standardization, plan and mutual adjustment, to stylized categorizations of dependencies such as pooled, sequential and reciprocal. Mintzberg (1979) took an organizational-level perspective and argued that specific coordination mechanisms are properties of particular kinds of organizations and environments. Finally, Crowston (1991) developed a typology of coordination problems to catalog coordination mechanisms that address specific types of interdependencies.

All those perspectives present high level and stylized categorizations of dependencies which are a powerful instrument in research endeavors. However, the various coordination mechanisms, and for that matter the classifications of dependencies, do not address the compound and dynamic nature of several types of tasks such as the design and development of complex physical and software systems. The collaborative tools of the future need to go step further and assess the characteristics of the task and assist the users in identifying and dealing with dependencies unknown a priori or that emerged as a consequence of the evolving characteristics of tasks. In the following section, we present a fine-grain view of congruence that focuses exclusively in the dynamic existence of interdependencies among tasks and their relationship with actors, independent of the properties of the those interdependent relationships.

2.1 A Model of Congruence

Given a particular set of dependencies among tasks, we are interested in identifying which set of individuals should be communicating and coordinating their activities and understand whether that occurs or not. The various dependency and communication relationships can be represented in matrix form. For instance, assigning individuals to particular work items can be represented by a people by task matrix where a one in cell *ij* indicates that worker *i* is assigned to task *j*. We will refer to this matrix as A. Following the same approach, we can think of a set of dependencies among tasks as a square matrix, named D, where a cell *ij* (or cell *ji*) indicates that task *i* and task *j* are interdependent. Now, if we multiply the matrices A and D, we obtain a people by task matrix, named AD, that represents the set of tasks a particular worker should be aware of, given the work items the person is responsible for and the dependencies of those work items with other tasks.

Then, we can obtain a representation of the coordination requirements among the different workers by multiplying the matrix AD by the transpose of the matrix A. This product results in a people by people matrix, named C_R. Finally, congruence can be determined by comparing the C_R matrix with a communication matrix, named C_A, that represents the interactions workers engaged in through different means of communication. Then,

given a particular set of dependencies among tasks, congruence is the proportion of interactions that actually occurred (given by C_A) relative to the total number of interactions that should have taken place (given by C_R). In other words, congruence represents the ratio of coordination requirements that were satisfied.

2.2 Research Questions

In this study, we present a technique to measure how congruent the actual organizational communication channels are relative to the theoretical interaction requirement imposed by the tasks. Specifically, we address the following research questions:

What are the consequences of congruence? The theoretical developments in the organizational literature suggest that congruence is an important factor affecting task performance. Consequently, we expect to find higher levels of performance associated with higher levels of congruence.

How does congruence come about? Numerous factors such as the attributes of the task and individual-level characteristics drive communication patterns. As these factors evolve over time, it is crucial to understand the impact on the development of congruence.

3. METHOD

There are several characteristics of large software development projects that make them ideal settings for studying coordination and the role of congruence. First, the development of complex software systems is carried out by a large number of individuals, grouped into teams, working in parallel on different components of the same software product. Furthermore, no single developer or a small group has the ability to create or even fully understand large and complex systems in their entirety (Kraut & Streeter, 1995). Then, the efforts of those interdependent units need to be tightly coordinated in order to successfully develop such systems. Secondly, software design and development activities produce large amounts of archival information that allow us to reconstruct the details of how activities depend among themselves, how those dependencies change over time as well as how developers interact. The availability of such rich source of data makes software development a more attractive research setting relative to other types of intellectual and interdependent tasks.

We collected data from a software development project of a large distributed system produced by a company that operates in the data storage industry. The data covered a period of almost three years of development activity. The company had one hundred and fourteen developers grouped into eight development teams distributed across three laboratories. All the developers worked full time into the project during the time period covered by our data. This setting provides a way to examine congruence as it relates to formal organizational structures as well as to geographical locations. We will refer to these two measures of congruence as structural and geographical congruence. Developers also use tools such as Internet Relay Chat (IRC) and a modification request (MR) tracking system to interact and coordinate their work. We built congruence measures based on these additional sources of data and we refer to them as IRC and MR communication congruence.

We performed two different studies. In the first study, we examined the role of congruence on task performance. In the second study, we examined the evolution of congruence between coordination requirements and actual communication over time. The following sections provide the details of each study and their results.

3.1 Study 1: The Impact of Congruence on Task Performance

The modification requests constitute our unit of analysis. Software development involves making a set of technical decisions that result in modifications to parts of the software. In order for the software to function correctly, the technical decisions made by the various developers must be compatible, therefore, coordination is required. Empirical research has shown that difficulties in communication and coordination breakdowns are recurring problems in software development (Curtis et al, 1988; Herbsleb & Mockus, 2003; Kraut & Streeter, 1995), particularly, when the work items are geographically distributed (Herbsleb & Mockus, 2003) and the task involves more than one team (Curtis et al, 1988; Kraut & Streeter, 1995). Then, we focus our analysis on the set of modification requests that involve individuals from different teams in the organization.

3.1.1 Description of the measures

The literature has identified a number of factors that affect development time and, consequently, the resolution of modification requests. Some of those factors are related to characteristics of the task such as the amount of code to be written and the priority of the task, whereas other factors capture relevant attributes of the individual developers and the teams that participate in the development task. In the following paragraphs, we first describe our dependent variable, resolution time of modification requests. Then, the independent variables of our model are presented. Finally, we describe a number of control measures that have also included in our model. Tables 1 and 2 summarize the descriptive statistics and the pair-wise correlations of the variables.

Task performance is measured as the time that took to resolve a particular modification request and accounts for all the time that the MR was assigned to developers. The modification requests reports contain records of when the MR was opened and resolved as well as every time the MR was assigned to a particular developer. Given this information, we can compute the amount of time that developers were actually working on the task. We acknowledge that some modification request may have longer resolution times because people are working on multiple MRs, or because a modification request was temporarily suspended to address other emergency work. We address these concerns with several control variables described later in this section. Descriptive statistics and an inspection of the Q-Q plot showed that the resolution time measure was highly skewed to the left and the log transformation provided the best approximation to a normal distribution. Then, our dependent variable is the logarithm of the actual resolution time.

Table 1: Descriptive Statistics (N=809).

	Mean	SD	Min	Max	Skew	Kurtosis[1]
Resolution Time (log)	0.888	1.297	0	6.490	1,476	1.350
Structural Congruence	0.552	0.210	0.239	0.982	0.276	-1.355
Geographical Congruence	0.763	0.124	0.461	0.954	-0.348	-0.420
MR Congruence	0.512	0.294	0.101	0.982	-0.144	-1.566
IRC Congruence	0.471	0.267	0.084	0.982	0.079	-1.279
Dependency	0.170	0.257	0	1	3.344	9.185
Priority	3.193	0.844	1	5	-1.026	0.784
Re-assignment	3.270	1.498	0	6	-1.034	0.463
Customer MR	0.120	0.132	0	1	7.308	15.130
Time	69.133	7.678	50	81	-0.370	-0.671
Change Size (log)	0.508	1.065	0	4.741	0.584	-0.180
Team Load	22.742	13.931	1.578	58.800	0.638	-0.104
Programming Experience	2	22	6.402	4.311	1.012	1.388
Tenure	25.488	18.581	1	76	-0.160	-0.657
Component Experience (log)	3.026	1.145	0	5.601	-0.979	-0.503

In order to compute the various measures of congruence, our independent variables, we need to build the matrices C_A and C_R described earlier. We used four communication paths to construct the data in the matrix C_A. The opportunities of interaction that exist for working in the same formal team or for working in the same location represent two potential paths of communication. The actual interactions among developers that occurred in the various IRC channels and in the modification requests reports provide alternative data for building a C_A matrix. The data for constructing C_R is extracted from the modification request reports. A modification request constitutes a change to the software such as adding a particular functionality or fixing an existing problem. Since a software system is a collection of files, a modification request might involve changes to one or more of those files. Moreover, the work can be performed by one or more developers. Then, a modification request provides us with a "developer *i* modified file *j*" relationship that constitutes our matrix A. In addition, we consider the technical decisions about the change to one file in a MR as dependent on the decisions made about changes to the other files involved in implementing the MR. Then, files changed together in a MR represent our dependency matrix D. The product of the matrices A, D and the transpose of the matrix A then give us the coordination requirements matrix C_R. The details of how each measure of congruence was computed are described in the following paragraph.

Table 2: Pair-wise correlations among the independent variables (N=809,* p < 0.05).

[1] The statistical package used centers the Kurtosis measure at 0.

		1	2	3	4	5	6	7
1	*Dependency*	-						
2	*Priority*	-0.043	-					
3	*Re-assignment*	0.061	-0.157*	-				
4	*Customer MR*	-0.003	0.087	0.042	-			
5	*Time*	-0.038	0.117	0.024	0.097	-		
6	*Change Size*	0.160*	-0.060	0.109	-0.037	-0.139	-	
7	*Team Load*	-0.082	-0.040	0.026	-0.005	0.122	-0.005	-
8	*Programming Experience*	-0.042	0.023	-0.100	-0.056	0.045	0.018	-0.129*
9	*Tenure*	-0.098	0.006	-0.005	-0.027	0.075	0.068	0.137*
10	*Component Experience*	0.051	-0.070	0.072	0.026	0.053	0.152	-0.005
11	*Structural Congruence*	-0.031	-0.032	0.006	-0.013	0.101	0.088	0.138*
12	*Geographical Congruence*	-0.015	-0.059	0.013	0.001	0.087	0.076	0.117
13	*MR Congruence*	-0.003	-0.063	0.013	0.001	0.175*	0.122	0.108
14	*IRC Congruence*	0.127	0.068	0.019	0.057	0.187*	-0.036	0.053

		8	9	10	11	12	13	14
9	*Programming Experience*	-						
10	*Tenure*	0.152*	-					
11	*Component Experience*	0.157*	0.189*	-				
12	*Structural Congruence*	0.083	0.069	0.087	-			
13	*Geographical Congruence*	0.086	0.107	0.057	-0.060	-		
14	*MR Congruence*	0.101	0.120	0.039	0.039	-0.032	-	
15	*IRC Congruence*	0.008	0.044	0.012	-0.027	0.093	0.150*	-

Structural Congruence captures the potential paths of communication that members of a formal team have through various mechanisms such as team meetings and other work-related activities. We build a communication network where an interaction between engineers *i* and *j* exists if they belong to the same formal team. *Geographical congruence*, similarly to the case of organization structure, is built around the idea of potential paths of communication that exists when individuals work in the same physical location (Allen, 1977; Olson & Olson, 2000). Then, in terms of the matrix of interactions, engineers *i* and *j* have a communication linkage if they work in the same location. Higher levels of congruence would mean that the geographic location of people matches their coordination needs so that relatively little coordination is required across sites. *MR communication congruence* considers an interaction between engineers *i* and *j* only when both *i* and *j* explicitly commented in the MR report. Multiple modification requests might refer to the same problem and later be marked as duplicates of a particular modification request. All duplicates of the focal MR were also used to capture the interactions among developers. The MR tracking system sends email to all the individuals listed in a CC list. We did not consider part of the communication network those recipient of email updates that did not explicitly contributed information to the modification report. Hence, our approach creates a network restricted to actual exchanges of information.

Finally, *IRC communication congruence* was computed based on interaction between developers from the IRC logs. Three raters, blind to the research questions, examined the IRC logs corresponding to the period of time associated with each MR and established an interaction between engineers *i* and *j* if they made reference to the bug ID or to the task or problem represented by the MR in their conversations. IRC communication was coded for 809 modification requests out of the 1983 MRs that compose our dataset. In order to assess the reliability of the raters' work, 10% of the MRs where coded by all raters. Comparisons of the obtained networks showed that 97.5% (79 out of 81) of the networks had the same set of nodes and edges.

Past research has proposed several additional factors that have an important impact on development time (Espinosa, 2002; Herbsleb & Mockus, 2003; Kraut & Streeter, 1995). We collected a number of control variables that are task-specific measures as well as team- and individual-level controls. Several task-specific factors such as task dependency, priority and task re-assignments could have an effect on development time. *Dependency* was measured as the number of modification requests that the focal MR depends on in order to for the task to be performed. Management prioritized the activities of the developer by using a scale from 1 to 5 in the modification request report where 5 was the highest priority and 1 the lowest. Then, that constituted our measure of *priority* of the MR. *Task re-assignment* was measured as the number of times an MR was re-assigned to a different engineer or team. This is particularly important because the new set of individuals need to build contextual information about the task that will impact the resolution of the MR. In addition, MRs opened by customers could represent work items with higher importance consequently affecting the resolution time. A dummy variable was used to indicate if the MR is associated with the service request from a customer. Finally, a time variable, measured in months from the beginning of the project, was added to control for efficiencies that might develop overtime and, consequently, affect the resolution time of modification requests.

The amount of code written or changed is a proxy for the actual development work done. The *change size* was computed as the number of files that were modified as part of the fix for the focal MR. Prior research (Espinosa, 2002) has used lines of code changed as a measure of the size of the modification, however, our evaluation showed equivalent results to those obtained with our measure of change size. Therefore, the results presented in this paper 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.

An experienced software engineer familiar with tools and programming languages can be more productive that inexperience developers (Brooks, 1995; Curtis et al, 1988; Robilliard et al, 2004). Furthermore, experience with the domain area and the application being developed helps accelerate development time (Curtis et al, 1988). We used archival information to compute several individual level measures of experience. First, *programming experience* was computed as the average number of years of programming experience prior to joining the company of all the engineers involved in the modification request. *Tenure* was measured as the average number of

months in the company of all the engineers that worked in the modification request at the time the work associated with the MR was completed. *Component experience* was computed as the average number of times that the engineers responsible for the MR have worked on the same files affected by the focal modification request. This measure was also log-transformed to satisfy normality requirements. Finally, *Team load* is a measure of the average work load of the team of engineers responsible for the components associated with the MR. This control variable was computed as the ratio of the average number of MRs in *open* or *assigned* state over the number of engineers in the groups during the time period that the MR was in *assigned* state.

3.1.2 Results

We performed several linear regression analyses to assess the effect of the congruence measures. The results are presented in Table 3. The results from Model I are consistent with previous research in software engineering (Espinosa, 2002; Herbsleb & Mockus, 2003). Factors such as the size of the modification, domain familiarity, and general programming experience are significant elements that affect resolution time of MRs.

Table 3: Results for MRs from the Random Sample Dataset (* p < 0.05, ** p < 0.01).

	Model I	Model II	Model III	Model IV	Model V	Model VI	Model VII
(Constant)	7.688**	7.973**	8.127**	8.032**	8.193**	7.572**	7.981**
Dependency	0.870*	0.892*	0.903*	0.891*	0.891*	0.859*	0.849*
Priority	-0.780*	-0.780*	-0.781*	-0.774*	-0.778*	-0.734*	-0.763*
Re-assignment	0.219*	0.221*	0.221*	0.221*	0.217*	0.218*	0.221*
Customer MR	-0.780	-0.782	-0.782	-0.764	-0.764	-0.78	-0.764
Time	-0.075*	-0.074*	-0.074*	-0.075*	-0.075*	-0.073*	-0.071*
Change Size	1.546*	1.552*	1.557*	1.571*	1.562*	1.417*	1.444*
Team Load	0.311*	0.313*	0.317*	0.323*	0.319*	0.307*	0.309*
Programming Exp.	-0.135*	-0.134*	-0.133*	-0.134*	-0.131*	-0.138*	-0.134*
Tenure	-0.271*	-0.275*	-0.276*	-0.272*	-0.274*	-0.268*	-0.267*
Component Exp.	-0.190*	-0.204*	-0.204*	-0.215*	-0.212*	-0.205*	-0.204*
Structural Congruence		***-0.490****	***-0.489****	***-0.475****	***-0.426****		***-0.473****
Geographical Congruence			***-0.293****	***-0.302****	***-0.295****		***-0.302****
MR Congruence				***-0.135****	***-0.117****		***-0.135****
IRC Congruence					***-0.096****		---
N	809	809	809	809	809	1983	1983
Adjusted R^2	0.791	0.812	0.827	0.841	0.859	0.753	0.842

Task-specific characteristics such as dependencies with other modification requests, the priority of the task as well as re-assignments of task responsibility increased development time. Models II though V introduce our measures of congruence in the analysis. In those models, the number of modification request in the sample is limited by the availability of the coded IRC interactions to construct the communication congruence measure. Models VI and VII show the results of the regression analysis performed against the entire dataset. Overall, the models show consistent statistically significant effects on all the congruence measures. The estimated coefficients of the congruence measures have negative values which are associated with a reduction in resolution time. Then, the results highlight the important role of congruence on task performance as well as the complementary nature of all communication paths. Table 4 presents a similar analysis broken down on a per product release basis.

The results in table 4 show that the effects of the various types of congruence differed across releases. Structural

Table 4: Results for MRs Associated with each Release (* p < 0.05, ** p < 0.01).

	Release 1		Release 2		Release 2	
	Model I	Model II	Model I	Model II	Model I	Model II
(Constant)	6.997**	7.107**	7.493**	7.653**	8.195**	8.318**
Dependency with other MRs	0.987*	0.969*	0.842*	0.802*	0.847*	0.893*
Priority	-0.567*	-0.553*	-0.678*	-0.665*	-0.709*	-0.709*
Re-assignment	0.341*	0.341*	0.218*	0.249*	0.232*	0.232*
Customer MR	-0.876	-0.884	-0.797	-0.746	-0.891	-0.891
Time	-0.081**	-0.082**	-0.069**	-0.067**	-0.071**	-0.069**
Change Size	1.417*	1.593*	1.509*	1.581*	1.387*	1.451*
Team Load	0.293**	0.299**	0.302*	0.302*	0.302*	0.307*
Programming Experience	-0.175*	-0.171*	-0.146*	-0.142*	-0.138*	-0.139*
Tenure	-0.245*	-0.237*	-0.271*	-0.265*	-0.288*	-0.279*
Component Experience	-0.319*	-0.305*	-0.278*	-0.281*	-0.232*	-0.256*
Structural Congruence		***-0.389****		***-0.402****		***-0.457***
Geographical Congruence		***-0.242****		***-0.307****		***-0.281****
MR Congruence		***-0.213****		***-0.183****		***-0.119****
IRC Congruence		***-0.115****		***-0.167****		***-0.089****
N	437	437	185	185	187	187
Adjusted R^2	0.813	0.884	0.802	0.858	0.794	0.869

congruence is associated with shorter development times in release 1 and 2 but its effect eroded by the third release. One possible interpretation is that the software evolved and new coordination requirements were introduced, consequently, the formal channels of communication provided by the organizational structure no longer satisfied them. In order to assess whether personnel turnover and mobility across teams contributed to this result, we examined archival data collected from the company. This analysis indicated a yearly turnover rate of only 3% and an inter-group mobility rate of less than 1%. Therefore, membership changes in the groups and

departing developers are unlikely to be substantial reasons for the lack of effect of structural congruence on resolution time. Geographical congruence had a positive effect on resolution time in all three releases, consistent with past research showing the detrimental effects of distance on communication (see Herbsleb & Mockus, 2003 and Olson & Olson, 2000 for reviews). Finally, communication congruence based on the interactions amongst engineers through the MR reports affected development time differently across releases, gaining statistical significance only in the last two releases. The lack of effect in the first release could be a result of engineers not using the modification request reports for communication in the early stages of development. Information seeking activity could have taken place face-to-face in informal meetings or planned group activities. The positive effect of structural congruence in the first release is consistent with this argument.

3.2 Study 2: Evolution of Congruence over Time

The analysis from study 1 showed that when communication amongst individuals matches the communication requirements imposed by the dependencies of the tasks, task performance is improved. In this study, we explore the evolution of the measures of congruence over time and examine how the patterns of congruence relate to individuals' performance in the project.

The data used in this analysis consisted of 809 modification requests used in the first study plus 167 modification request from the forth release of the product. This last group of modification request was not included in the analysis reported in study 1 because we were not able to obtain data for some of the control variables. We did not require these control variables for Study 2, so we were able to include data from all four releases of the product. One difficulty when doing a longitudinal analysis of a software project is the changing nature of the tasks. For instance, in the first release, an important amount of feature development activity took place during the period of analysis. By the third and fourth releases, the modification requests were mostly related to defect resolution. Therefore, we also explore the relationship between the characteristics of the task, feature development or defect resolution, and the evolution of the congruence measures.

Since we are using the software code history to compute communication requirements (C_R), our congruence measures are most relevant to tasks focused on the software code, rather than, say, analysis or design. In order to achieve this focus, we restricted our analysis to modification requests that were resolved between the day of "code-freeze"[2] until the day the product was released. These changes are both fixes for defects encountered during the Quality-Assurance testing phase, and any feature work that was not finalized before the code freeze date. We selected these data to clearly identify a period of time where there is a high level of company-wide focus on programming tasks.

[2] "code-freeze" is a term used in software development to indicate the beginning of the development phase that focuses on getting the product to the levels of quality required for a first-customer-shipment.

As in study 1, we computed four different measures of congruence based on organizational structure, geographical location, interactions captured on IRC and in the MR reports. The communication networks were built on a weekly basis. Congruence was computed comparing the communication requirements matrix from week t_n to the actual communication matrix from week t_{n-1}, because we assume that developers would discuss a particular problem before making the actual changes in the source code[3]. The communication network based on IRC or modification requests represents an aggregate measure across all MRs resolved in a particular week.

Our analysis showed that the different measures of congruence varied significantly across releases. Figure 1 shows the average level of each measure of congruence across the different releases. In the first release, structural and geographical congruence dominate while communication congruence based on MRs or IRC are almost absent. In later releases, structural congruence decreases significantly, particularly in the third and forth releases. This result is consistent with the results found in study 1, suggesting that the dependencies among the various components of the software system are changing over time and the work requires the contribution of individuals from different teams. The measures of communication congruence based on MR and IRC increase in release 2 and they remain high during the last two releases. The increase in communication congruence coincides with the gradual decrease of structural congruence. A possible interpretation of this result is that developers are learning to substitute the lack of formal communication paths with interactions through other means such as IRC and MR reports.

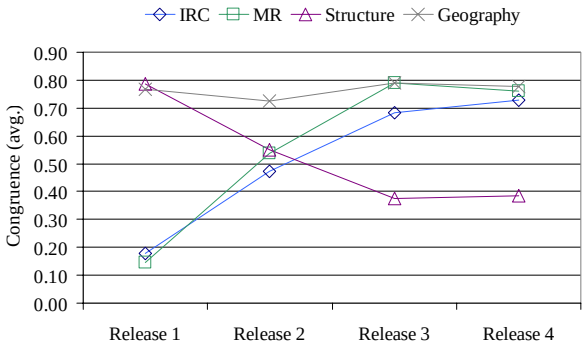

Figure 1: Evolution of the Congruence Measures across Releases.

We also examined the evolution of the congruence measures from a statistical point of view using a repeated measures type of analysis. Table 5 shows a significant effect of time on the various types of congruence measures. Moreover, the interaction of time and type of congruence is significant suggesting that the various measures are changing over time in different ways as shown in figure 1. Although whether the modification request refers to a

[3] We also computed the congruence measures using week t_n for both required and actual communications, and the trends remained the same.

feature development or a defect task affects the levels of congruence, that effect remains the same overtime, hence, the lack of "*Time-Type of Task*" interaction effect.

Table 5: Effect of Time on the Type of Congruence Measure.

	F	df	p
Time	107.028	3	**<0.001**
Type of Congruence	112.208	3	**<0.001**
Type of Task	8.465	1	**0.004**
*Time * Type of Congruence*	116.051	9	**<0.001**
*Time * Type of Task*	0.387	3	0.742

Mockus et al (2002) reported that in open source and commercial projects most of the modifications to the software are made by a small number of developers. These findings provide a useful framework to construct classify developers based on their contributions and examined the pattern of interaction and, consequently, patterns of congruence of different groups of developers. We computed the contributions of the developers and we encountered that 50% of the system was built by only 18 developers (15.6%). We, then, separated the developers and their interactions into two groups and repeated the analysis reported above. Figure 2 shows the evolution of congruence for the top 18 contributors across releases. The general patterns are similar to the overall results shown in figure 1.

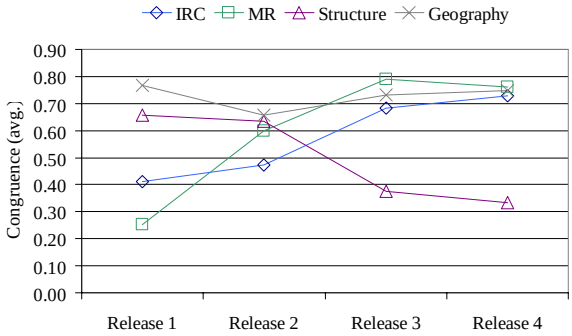

Figure 2: Congruence Measures across Releases based on Top Contributors Interactions.

Figure 3 depicts a very different result for the rest of the developers. These developers do not seem to use the computer-mediated communication means to interact with the right set of people. Consequently, they never achieve high levels of congruence in the IRC and MR congruence measures.

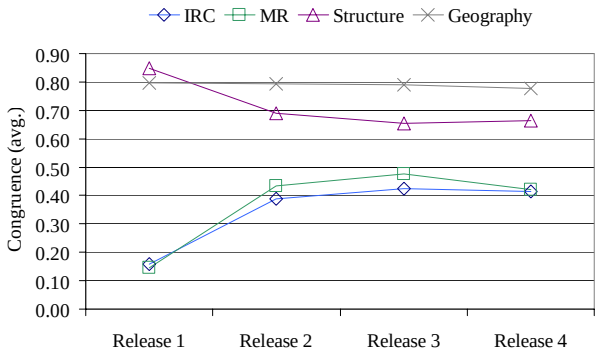

Figure 3: Congruence Measures across Releases for the rest of the developers.

There are many factors that can lead to the differences in interaction and coordination patterns reported in figures 2 and 3. Table 6 indicates that the two groups of developers do not differ in terms of programming experience, domain experience, their level of education and their tenure in the company. A possible source of different, then, could be related to the top contributors' structural position in the communication networks.

Table 6: Differences between developers' populations.

	t	df	p
Programming Experience	-1.85	30	0.073
Domain Experience	-0.59	23	0.556
Graduate Education	1.03	23	0.311
Tenure in the company	1.21	23	0.239

4. IMPLICATIONS FOR COLLABORATIVE TOOLS DESIGN

The results presented in the previous section provide interesting insights about the patterns of communications and coordination among individuals working on tasks with dynamic sets of interdependencies. Collaboration and awareness tools can benefit from the technique proposed in this study in various ways. Once a measure of dependency is available, a tool could assess the congruence between coordination requirements and communication at any point in time. Then, managers or other stakeholders can take, if required, the necessary steps to facilitate the appropriate flows of communication. For instance, in a study of communication and coordination in a jet engine design project, Sosa et al. (2004), highlighting the difficulty of managing cross-boundary interdependencies, provide examples of interdependent teams that did not interact at all among them. The lack of appropriate communication resulted in difficulties at the time of integrating the various subsystems. Project management tools could use congruence type of information to provide a signaling mechanism that alerts project managers of potential issues with certain tasks prior to important milestones such as system integration.

In the context of large and complex projects, where tens or hundreds of individuals participate, identifying the right person to interact with and coordinate interdependent activities might not be a straight forward task. The congruence measures could provide the backend to augment an awareness tool that provides real-time information of the likely set of workers that a particular individual might need to communicate with. In software development, for instance, tools such as integrated development environments (IDE) could use the congruence measures to recommend a dynamic "buddy list" every time particular parts of the software are modified. In this way, the developer becomes aware of the set of engineers that modified parts of the system that are interdependent with the one the developer is working on. In the context of collaboration and awareness in software development, tools such as TUKAN (Schummer & Haake, 2001) and Palantir (Sarma et al, 2003) have been proposed. This set of applications provides visualization and awareness mechanisms to aid developers identify and handle modification to the same software artifacts such as source code files. The technique proposed in this paper goes beyond the identification of artifacts shared or modified by multiple developers but also would allow developers identify those files that have dependencies among them when those dependencies are not explicitly determined. Therefore, tools such as TUKAN and Palantir would be enhanced by integrating the congruence measures.

In sum, collaboration technologies could use the congruence information in a variety of ways to understand what needs to be brought into the user experience.

5. REFERENCES

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

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

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

Crowston, K.C. (1991). *Toward a Coordination Cookbook: Recipes for Multi-Agent Action.* Ph.D. Dissertation, Sloan School of Management, MIT.

Curtis, B., Kransner, H. and Iscoe, N. (1988). A field study of software design process for large systems. *Comm. ACM,* 31, 11 (Nov.), 1268-1287.

Espinosa, J.A. (2002). *Shared Mental Models and Coordination in Large-Scale, Distributed Software Development*. Ph.D. Dissertation, Graduate School of Industrial Administration, Carnegie Mellon University.

Herbsleb, J.D. and Mockus, A. (2003). An Empirical Study of Speed and Communication in Globally Distributed Software Development. *IEEE Trans. on Soft. Eng.*, 29, 6 (Jun.), 481-494.

Kraut, R.E. and Streeter, L.A. (1995) Coordination in Software Development. *Comm. ACM,* 38, 3 (Mar.), 69-81.

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

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

Mockus, A., Fielding, R. and Herbsleb, J.D. (2002). Two Case Studies of Open Source Software Development: Apache and Mozilla. *ACM Trans. on Soft. Eng. and Method.*, 11, 3 (Jul.), 309-346.

Olson, G.M. and Olson, J.S. (2000). Distance Matters. *Human-Computer Interaction*, 15, 2 & 3, 139-178.

Robillard, M.P, Coelho, W. and Murphy, G.C. (2004). How Effective Developers Investigate Source Code: An Exploratory Study. *IEEE Trans. on Soft. Eng.*, 30, 12 (Dec.), 889-903.

Sarma, A., Noroozi, Z., and van der Hoek, A. (2003). Palantir: Raising Awareness among Configuration Management Workspaces. *In Proceedings of the International Conference in Software Engineering (ICSE '03)*, Portland, Oregon, 444-454.

Schummer, T. and Haake, J.M. (2001). Supporting Distributed Software Development by Modes of Collaboration. *In Proceedings of the Seventh European Conference on Computer Supported Cooperative Work (ICSE '03)*, 79-98.

Sosa, M.E., Eppinger, S.D. and Rowles, C.M. (2004). The misalignment of product architecture and organizational structure in complex product development. *Management Science*, 50, 12 (Dec.), 1674-1689.

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