Identification of Coordination Requirements: 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
Task dependencies drive the need to coordinate work activities. We describe a technique for using automatically generated archival data to compute coordination requirements, i.e., who must coordinate with whom to get the work done. Analysis of data from a large software development project revealed that coordination requirements were highly volatile, and frequently extended beyond team boundaries. Congruence between coordination requirements and coordination activities shortened development time. Developers, particularly the most productive ones, changed their use of electronic communication media over time, achieving higher congruence. We discuss practical implications of our technique for the design of collaborative and awareness tools.

Categories and Subject Descriptors
H.5.3 [**Information Interfaces and Presentation**]: Groups and Organization Interfaces – *collaborative computing, computer-supported cooperative work, organizational design.*

General Terms
Management, Performance, Human Factors.

Keywords
Coordination, Collaboration tools, Task Awareness Tools, Dynamic Network Analysis.

1. INTRODUCTION
It has long been observed that organizations carry out complex tasks by dividing them into smaller interdependent work units assigned to groups and coordination arises as a response to those interdependent activities [21]. Communication channels emerge in the formal and informal organizations. Over time, those information conduits develop around the interactions that are most critical to the organization's main task [12]. This is particularly important in product development organizations which organize themselves around their products' architectures because the main components of their products define the organization's key subtasks [30]. Organizations also develop filters that identify the most relevant information pertinent to the task at hand [9].

Changes in task dependencies, however, jeopardize the appropriateness of the information flows and filters and can disrupt the organization's ability to coordinate effectively. For example, Henderson & Clark [15] found that minor changes in product architecture can generate substantial changes in task dependencies, and can have drastic consequences for the organizations' ability to coordinate work. If we had effective ways of identifying detailed task dependencies and tracking their changes over time, we would be in a much better position to design collaborative and task awareness tools that could help to align information flow with task dependencies.

Identifying task dependencies and determining the appropriate coordination mechanism to address the dependencies is not a trivial problem. Coordination is a recurrent topic in the organizational theory literature and, as we will discuss in the next section, many stylized types of task dependencies and coordination mechanisms have been proposed over the past several decades. However, numerous types of work, in particular non-routine knowledge-intensive activities, are potentially full of fine-grain dependencies that might change on a daily or hourly basis. Conventional coordination mechanisms like standard operating procedures or routines would have very limited applicability in these dynamic contexts. Therefore, designing tools that support rapidly shifting coordination needs requires a more fine-grained level of analysis than what the traditional views of coordination provide.

In this paper, we develop a technique to measure task dependencies among people, and the "fit" between these task dependencies and the coordination activities performed by individuals. We refer to the fit measure as congruence. Using data from a software development project, we found that patterns of task dependencies among people are in fact quite volatile, confirming our suspicion that a fine-grained view of dependencies is important. We then explored the consequences of congruence for the speed and efficiency of work. 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 communica-

CSCW'06, November 4–8, 2006, Banff, Alberta, Canada.

tion 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 traditional views of coordination need to be extended to account for the volatile coordination needs of non-routine intellectual work. Further, it has the practical implication 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, we show how coordination requirements can be computed automatically, given a person's activities, to determine the set of people he or she should be aware of or communicate with. 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 [4]. The definition or selection of an organizational structure is influenced, among several factors, by the interdependencies among tasks. March and Simon [21] argued that coordination encompasses more than just a traditional division of labor and assignment of tasks. They proposed numerous mechanisms such the division of the task into nearly independent parts. They also argued that schedules and feedback mechanisms are required when interdependence is unavoidable. Thompson [29] 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 [22] took an organizational-level perspective and argued that specific coordination mechanisms are properties of particular kinds of organizations and environments. Finally, Crowston [6] 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 useful in the context of enduring structures. It is not surprising, then, that the organizational design field has devoted a significant amount of attention to these types of coordination mechanisms. However, the various coordination mechanisms, and for that matter the classifications of dependencies, do not address the dynamic nature of tasks such as non-routine intellectual work. Failure to identify the new needs for coordination and information exchange might hinder the organization's ability to adapt to changes in their competitive environment [15].

When members of a team are physically collocated and coordination requirements within the team change, there are numerous ways for team members to identify the new needs and act on them such as group and status meetings and managerial intervention. However, social and communicational barriers pose important obstacles for coordination among individuals from different formal teams. Given these challenges, if the coordination requirements always involve the same set of developers, it is expected that over time individuals would develop common knowledge or a shared mental model that would reduce the possibility of coordination breakdowns [10]. Unfortunately, rapidly changing coordination requirements would represent a more demanding environment. Therefore, it is also important to understand how the coordination requirements differ over time. This discussion leads to our first research question:

RQ 1: How stable are coordination requirements?

The organizational literature suggests that congruence is an important factor affecting task performance [4, 5]. For instance, mismatch between interdependent design tasks and coordination might have impact on the quality of airplane engines [28]. Moreover, in software engineering, coordination breakdowns can lead to longer development times [10, 16] and higher number of defects and higher costs [7]. We expect to find higher levels of task performance associated with higher levels of congruence, leading to the following research question:

RQ 2: Is higher congruence associated with better task performance?

Numerous factors such as the attributes of the task and individual-level characteristics drive communication and coordination patterns. As these factors evolve over time, it is crucial to understand the impact on the development of congruence, raising our third research question:

RQ 3: How do various types of congruence change over time?

3. METHOD

3.1 Setting and data

Large software development projects building new systems consist of many non-routine intellectual tasks. There are several characteristics of such projects that make them good 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 [19]. 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 on each other, how those dependencies change over time, as well as how developers interact. The availability of such a rich source of data makes software development a very attractive research setting for studying coordination.

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 and the first four releases of the product. The company had one hundred and fourteen developers grouped into eight development teams distributed across three R&D laboratories. All the developers worked full time on 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 and geographical locations as well as computer-mediated communication channels.

Our unit of analysis is a modification request (MR) which can be thought of as a unit of work. An MR is a requested change to the

software such as adding a particular functionality or fixing an existing problem. 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. Consequently, some type of coordination is required. The opportunities of interaction that exist for working in the same formal team or for working in the same location represent two potential paths for coordinating the work. In our research setting, developers also use tools such as Internet Relay Chat (IRC) and the MR tracking system to interact and coordinate their work. For instance, the MR tracking system keeps track of the progress of the task, comments and observations made by developers as well as additional material used in the development process.

This paper presents the results of three analyses. First, we looked at the stability of coordination requirements over time (RQ 1). Similarly for research question 2, we examined the role of congruence in task performance. Finally, we examined the evolution of congruence between coordination requirements and actual communication over time (RQ 3). In the following sections, we discuss the dynamic nature of coordination requirements, our measures of congruence are described, and the details of each analysis and its results are presented.

3.2 Measuring Coordination Requirements

Given a particular set of dependencies among tasks, we are interested in identifying which set of individuals should be coordinating their activities. The various dependency 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 "*Task Assignments"*. Following the same approach, we can think of a set of dependencies among tasks as a square matrix where a cell *ij* (or cell *ji*) indicates that task *i* and task *j* are interdependent. We will refer to this matrix as *"Task Dependencies"*. Now, if we multiply the *Task Assignment* and *Task Dependencies* matrices, we obtain a people by task matrix 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. Finally, a representation of the coordination requirements among the different workers is obtained by multiplying the product of the *Task Assignment* and *Task Dependencies* matrices by the transpose of the *Task Assignment* matrix. This product results in a people by people matrix where a cell *ij* (or cell *ji*) indicates the extent to which person *i* works on tasks that share dependencies with the tasks worked on by person *j*. In other words, the resulting matrix represents the *Coordination Requirements* or the extent to which each pair of people needs to coordinate their work. In the analysis that follows, we use a dichotomized version of the *Coordination Requirements* matrix indicating, for each pair of people, whether or not a task dependency exists.

The data for building the *Coordination Requirements* matrix was extracted from the modification request reports. Since a software system is a collection of files, carrying out a requested modification involves 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 *Task Assignment* matrix. In addition, the technical decisions about the change to one file in a MR are ordinarily highly interdependent with the decisions made about changes to the other files involved in implementing that MR. Then, counting the number of times, over the life of the system, that two files were changed together in a single MR represents our *Task Dependency* matrix. Using those *Task Assignments* and *Task Dependencies* matrices, we obtain the *Coordination Requirement* matrix as described above.

3.3 Stability of the Coordination Requirements

In this section, we address our first research question by analyzing the evolution of the coordination requirements. We seek to assess whether the needs to coordinate, in fact, change over time and how rapidly they change. As we argued in section 2, we focused our attention in two key aspects of the coordination requirements: the overall rate of change in coordination need and the amount of coordination needed that crosses team boundaries. Figure 1 depicts the evolution of both factors on a weekly basis from the third release of the project. The rate of change in the coordination needs is computed comparing the *Coordination Requirements* matrices from week *t* against week *t-1*. Then, a 10% value of change in the coordination needs in week *t* means 10% of the coordination requirements of that week did not exist in the previous week (*t-1*). We observe that periods of stability are followed by several weeks of substantial change in the coordination requirements. The second, third and fourth releases presented similar volatile patterns, hence, the decision to present the data from only one release. The first release of the product showed lower levels of volatility in the rate of change as well as in the amount of external coordination requirements.

Figure 1 also shows the amount of coordination requirements that involved individuals from different teams in each week relative to the total amount of coordination needs. We observe that the need to coordinate with individuals outside the group as well as the set of individuals to coordinate with show substantial volatility week to week. In other words, the software repository data collected suggests that the task dependencies among people are quite dynamic and developers face constantly changing coordination requirements.

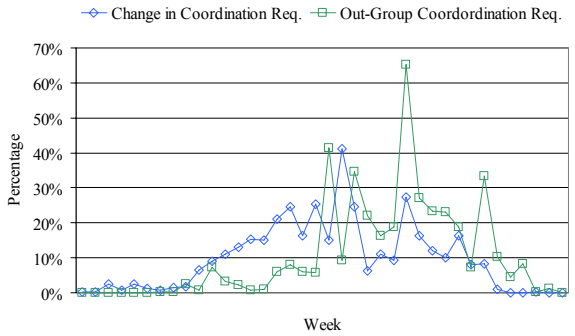

Figure 1: Example of the Evolution of Coordination Requirements from Week to Week.

3.4 Measuring Congruence

Congruence can be determined by comparing the *Coordination Requirements* matrix with an *Actual Coordination* matrix that represents the interactions workers engaged in through different means of coordination. Then, given a particular set of dependencies among tasks, congruence is the proportion of coordination activities that actually occurred (given by the *Actual Coordination* matrix) relative to the total number of coordination activities that should have taken place (given by the *Coordination Requirements* matrix). For example, if, for a particular modification request, the *Coordination Requirements* matrix shows that 10 pairs should coordinate, and of these, 5 show *Actual Coordination* interactions, then the congruence is 0.5. Congruence represents the proportion of coordination requirements that were satisfied, hence its value always falls between 0 and 1.

Our approach for measuring congruence builds on the idea of "fit" from the organizational theory literature. The measures of fit refer to the match between a particular organizational design and the organization's ability to carry out a task [4] and they have, traditionally, focused on two factors: the temporal dependencies among tasks that are assigned to organizational groups and the formal organizational structure as a means of communication and coordination [5, 20]. The method proposed in this paper extends the standard measures of fit by providing a finer-grain level of analysis and also assessing the role of multiple ways of coordinating activities.

In section 3.2, we discussed how the *Coordination Requirement* matrix was generated. In order to compute a measure of congruence, we also need to build the *Actual Coordination* matrix which represents the coordination activities that occurred. These activities could take numerous forms that might involve communication and exchange of information. The mere existence of communication does not necessarily mean that coordination is taking place. For instance, a chat session about last weekend's football results does not involve any type of coordination related to the tasks. Research on the use of chat in the workplace [17], however, and specifically chat as used by distributed software development teams [14], shows that the majority of exchanges are about the work. We believe that the amount of communication within a given text channel is a reasonable, if approximate, measure of coordination behaviors, such as project management or exchange of technical information.

We considered four coordination paths to construct our *Actual Coordination* matrices:

Structural Congruence captures 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. We build the actual coordination matrix where a coordination activity 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 and coordination that exist when individuals work in the same physical location [1, 24]. Then, in terms of the matrix of coordination activities, engineers *i* and *j* have a 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 exchange of technical information between engineers *i* and *j* only when both *i* and *j* explicitly commented in the modification request report. Multiple modification requests might refer to the same problem and later be marked as duplicates of a particular modification request. All duplicates of the focal MR were also used to capture the interactions among developers. Finally, *IRC communication congruence* was computed based on interaction between developers from the IRC logs. Three raters, blind to the research questions, examined the IRC logs corresponding to the period of time associated with each MR and established an interaction between engineers *i* and *j* if they made reference to the bug ID or to the task or problem represented by the MR in their conversations. In order to assess the reliability of the raters' work, 10% of the MRs where coded by all raters. Comparisons of the obtained networks showed that 97.5% of the networks had the same set of nodes and edges.

3.5 The Impact of Congruence on Task Performance

Empirical research has shown that difficulties in communication and coordination breakdowns are recurring problems in software development [7, 16, 19], particularly when the work items are geographically distributed [16] and the task involves more than one team [7, 19]. For these reasons, we focus our analysis on the set of modification requests that involve more than one software development team.

3.5.1 Description of the measures

The data were collected from the software repositories of the first three releases of the product. Software development is a multi-phase process that includes analysis of the problem, design and the actual implementation of the software. Software repositories do a very good job of capturing the activities that took place during the implementation phase. Therefore, we restricted our analysis to modification requests that were resolved between the day of "code-freeze" until the day the product was released. "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. 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. A total of 1983 multi-team modification requests were identified. A random sample of 809 MRs was selected to code the interactions amongst developers using Internet Relay Chat (IRC). The time consumed by the coding process is the only reason for not considering the entire 1983 MRs.

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, we describe a number of control measures that were also included in our model. Table 1 summarizes the descriptive statistics of the variables. The pair-wise correlations amongst all the variables in the model (dependent variable and predictors) were

Table 1: Descriptive Statistics (N=809).

	Mean	SD	Min	Max	Skew	Kurtosis[1]
Resolution Time (log)	1.888	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
Release	1.762	0.847	1	3	0.473	-1.446
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	6.402	4.311	2	22	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

computed. The highest correlation was 0.187 indicating the lack of collinearity issues in the models.

Our measure of task performance is *Resolution Time* which is measured as the time it 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 MR 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. For this reason, our dependent variable is the logarithm of the actual resolution time. The measures of congruence described in section 3.2 constitute our independent variables. Therefore, our analysis assesses the role of four different potential paths of communication: working in the same formal team, working in the same location, and interactions through two computer-mediated tools: IRC and the MR-tracking system.

Past research has proposed several additional factors that have an important impact on development time [10, 16, 19]. 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. This rating 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. Re-assignment impact resolution time because each new developer needs to build up contextual information about the task. 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, the *release* variable identifies the release of the product that the modification request is associated with. This variable could also be considered as a proxy for time to control for efficiencies that might develop over time and, consequently, affect the resolution time of modification requests.

The amount of code written or changed is a proxy for the actual amount of development work done. The *change size* was computed as the number of files that were modified as part of the change for the focal MR. Prior research [10] 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 substantially more productive than an inexperienced developer [3, 7, 25]. Furthermore, experience with the domain area and the technical characteristics of the application being developed helps accelerate development time [7]. We used archival information as well as data from the software repositories 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 modifica-

[1] We used a statistical package that centers the Kurtosis measure at 0.

Table 2: Results from OLS Regression of Effects on Task Performance ($^{+}$ p < 0.10, * p < 0.05, ** p < 0.01).

	Model I	Model II	Model III	Model IV
(Intercept)	2.987**	3.631**	1.572*	1.751*
Dependency	0.897*	0.653*	0.784*	0.712*
Priority	-0.741*	-0.681*	-0.702*	-0.712*
Re-assignment	0.423*	0.487*	0.304*	0.324*
Customer MR	-0.730	-0.821	-0.932	-0.903
Release	-0.154*	-0.137*	-0.109*	-0.098*
Change Size (log)	1.542*	1.591*	1.428*	1.692*
Team Load	0.307*	0.317*	0.356*	0.374*
Programming Experience	-0.062*	-0.162*	-0.117*	-0.103*
Tenure	-0.269*	-0.265*	-0.239*	-0.248*
Component Experience (log)	-0.143*	-0.143*	-0.195*	-0.213*
Structural Congruence		**-0.526***		**-0.483***
Geographical Congruence		**-0.317***		**-0.312***
MR Congruence		**-0.189***		**-0.129***
IRC Congruence		**-0.196***		**--**
Interaction: Release X Structural Congruence		**0.007**		**0.009**
Interaction: Release X Geographical Congruence		**-0.013**		**-0.017**
Interaction: Release X MR Congruence		**-0.009**$^{+}$		**-0.011**$^{+}$
Interaction: Release X IRC Congruence		**-0.017***		**--**
N	809	809	1983	1983
Adjusted R^2	0.787	0.872	0.756	0.854

tion 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 modification request 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 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.

3.5.2 Results

We performed several linear regression analyses to assess the effect of the congruence measures. The results are presented in Table 2. Models I and III are baseline regressions considering only the control factors. Models II and IV introduced our measures of congruence in the analysis. We also added interaction terms in the models to assess whether the role of congruence changes across the different releases of the product. The interaction terms are denoted "Release X Congruence" and they are computed, as usual, by multiplying the values of the two variables of interest, in this case *Release* and *Congruence*. The availability of the coded IRC data to construct the communication congruence measure was limited to 809 modification requests, therefore, models I and II show the results of the OLS regression on the restricted dataset. Model III and IV present the results of the analysis performed against the entire dataset. An examination of the residuals of the various models did not indicate any heteroscedasticity or over-fitting problems that would invalidate the results.

Consistent with previous empirical work in software engineering, factors such as the size of the modification, domain familiarity, and general programming experience are significant elements that affect resolution time of MRs [10, 16]. 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 and IV 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. The results highlight the important role of congruence on task performance as well as the complementary nature of all communication paths. Structural congruence is associated with shorter development times suggesting that when coordination requirements are contained within a formal team and appropriate communication paths exists, task performance increases. Geographical congruence had a positive effect on resolution time, consistent with past research that argued distance has detrimental effects on communication (see [16] and [24] for reviews). Communication congruence based on the interactions amongst engineers through the MR reports as well as IRC were also statistically significant suggesting the usefulness of these tools in facilitating coordination among individuals that belong to different teams and could potentially be geographically distributed. The results showed that the only statistically significant interaction term was Release X IRC congruence. The negative coefficient suggests that in later releases the effect of IRC

congruence on resolution time of modification requests is stronger.

The measures of structural and geographical congruence could be affected by personnel turnover and mobility across teams. In order to assess whether these factors contributed to the results, we examined archival data collected from the company and we determined a yearly turnover rate of only 3% and an inter-group mobility rate of less than 1%. We eliminated all the modification requests that involved the individuals that left the company or changed group membership. The results were in agreement with those reported in table 2.

3.6 Evolution of Congruence over Time

The previous analysis showed that when communication amongst individuals matches the communication requirements imposed by the task dependencies, task performance is improved. Next, 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 the 809 modification requests used in the previous analysis plus 167 modification requests from the fourth release of the product. This last group of MRs was not included in the previous analysis because we were not able to obtain data for some of the control variables. We did not require these control variables for examining the evolution of congruence, so we were able to include data from all release here.

In this analysis, we also computed the 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 *Coordination Requirements* matrix from week t_n to the *Actual Coordination* 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[2]. The communication network based on IRC or modification requests represents an aggregate measure across all MRs resolved in a particular week.

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, i.e. feature development or defect resolution, and the evolution of the congruence measures.

Our analysis showed that the different measures of congruence varied significantly across releases. Figure 2 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 our results on the volatility of coordination requirements, 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.

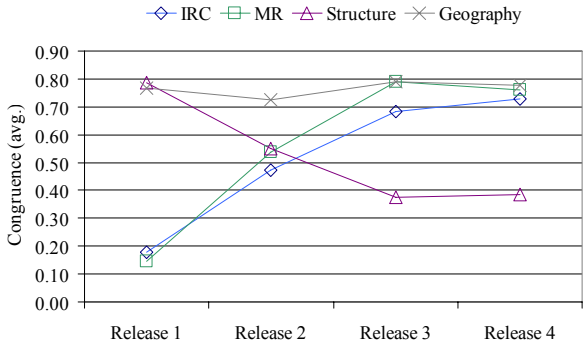

Figure 2: Evolution of the Congruence Measures across Releases.

We also examined the evolution of congruence from a statistical point of view using a repeated measures type of analysis. We considered congruence as the dependent variable and we considered the main effect of time, type of task and type of congruence measure as well as the interaction terms. Type of task refers to whether the modification request refers to a feature development or a defect task. Table 3 shows a significant main effect of time on congruence as well as the type of congruence. 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 2. Type of task has a main positive effect on congruence which is higher for feature development tasks. However, the effect of type of task remains the same overtime as suggested by the lack of significant in the "*Time-Type of Task*" interaction effect.

Table 3: Effect of Time on Congruence.

		F	p
Main Effects	*Time*	107.028	**<0.001**
	Type of Congruence	112.208	**<0.001**
	Type of Task	8.465	**0.004**
Interactions	*Time * Type of Congru.*	116.051	**<0.001**
	*Time * Type of Task*	0.387	0.742

Appropriate communication and coordination is an integral part of the software development process [16, 19]. The significant changes in communication patterns shown by figure 2 raise an interesting question: are all developers able to identify the changes in coordination requirements and adapt their communication paths accordingly? Mockus et al [23] 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 identify the most productive

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

developers, in order to compare their coordination behaviors with the less productive developers.

We computed the contributions of the developers and we observed that 50% of the modifications made to the software system were done by only 18 (15%) of the developers (see figure 3). We then separated the developers and their interactions into two groups and repeated the analysis reported above. Figure 4 shows the evolution of congruence for the top 18 contributors across releases. The general patterns are similar to the overall results shown in figure 2. Structural congruence decays over time while MR and IRC communication congruence increase considerably in the last two releases.

Figure 3: Proportion of Changes per Developer per Release.

On the other hand, Figure 5 depicts a very different result for the rest of the developers. Structural congruence decreases over time but not as drastically as in the case of the top performers. Moreover, 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.

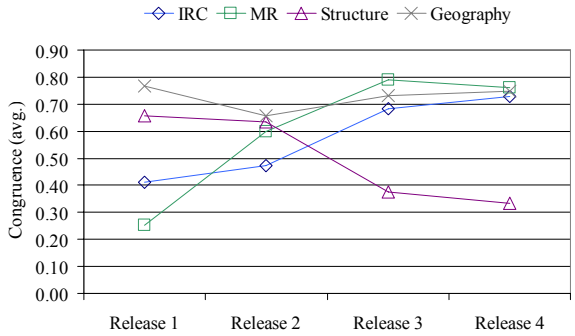

Figure 4: Congruence Measures across Releases based on Top Contributors Interactions.

The software engineering literature suggests that top developers typically have an order of magnitude better performance than average developers and the sources of that disparity are usually attributed to differences in experience and cognitive ability [7]. In our research setting we did not find evidence that differences in experience and familiarity with the system or attributes of the tasks were sources of difference in performance. Table 4 shows the results of comparing the other developers against the top performers. The two groups of individuals do not differ in terms of domain experience, their level of education and their tenure in the company. The disparity in programming experience is marginally significant, suggesting that the top performers might have slightly deeper programming experience than the rest of the developers. We also compared some of the attributes of the modifications requests that the two groups of developers worked on such as the average size of the changes made to the software, the average number of lines added to and removed from the software. The comparison of the average size of the modification request was marginally significant, suggesting that top performers tended to work on slightly larger changes to the software. However, there was no statistically significant difference in terms of lines of code added or deleted.

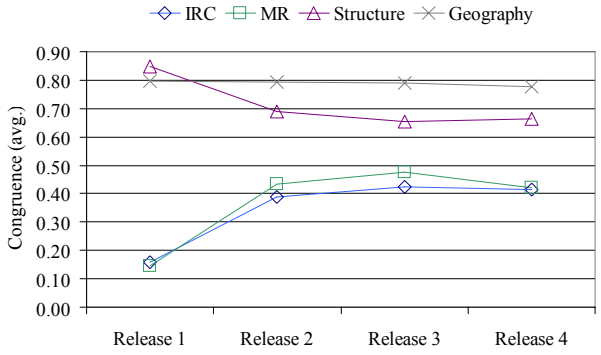

Figure 5: Congruence Measures across Releases for the Rest of the Developers.

Although we do not present any evidence of causality between patterns of communication and developer's performance, Figures 4 and 5 raise interesting questions that should be addressed in future research. Our results suggest that traditional thinking [6] may not capture all of the important attributes of top-performing developers, since their performance seems to have a substantial social component.

Table 4: Differences between developers' populations.

	t	p
Programming Experience	-1.85	0.073
Domain Experience	-0.59	0.556
Graduate Education	1.03	0.311
Tenure in the company	1.21	0.239
Avg. Size of Changes	-1.79	0.072
Avg. Lines Added	-1.51	0.148
Avg. Lines Deleted	0.95	0.341

4. DYNAMIC DEPENDENCIES AND DISTRIBUTED WORK

This paper presents a view of coordination that extends traditional conceptualizations of coordination by taking a fine-grain level of analysis to better examine the mismatches between dependencies and coordination activities. Those gaps could have major implications for the productivity and the quality of the output of product development organizations [7, 10, 16, 28] and for non-routine intellectual work more generally. Our empirical results suggest that the technique proposed here provides a useful framework to

examine the consequences of coordination requirements that are not satisfied.

The dynamic nature of dependencies that exist in complex tasks such as software development is another important factor addressed in our investigation. Individuals have difficulties identifying task interdependencies that are not obvious or explicit [28] and the workers' ability to recognize dependencies diminish as coordination requirements change over time [15]. For these reasons, volatility in the coordination requirements represents a major hurdle for work groups and, particularly, for those that are geographically distributed. Collaborative tools could play an important role in reducing the gap between recognized and actual interdependencies. It would be highly desirable for future tools to be able to 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. The technique proposed in this paper provides a framework for those future tools.

The work presented in this paper also has limitations that should be addressed in future research. Our study examined a software development organization dedicated to produce a single product. However, several organizational attributes (e.g. leadership styles, incentive mechanisms) and characteristics of the knowledge-intensive tasks could affect coordination and communication patterns among individuals or groups. A next logical step would be to examine a different organizational or task context. In addition, our analysis is correlational, hence, no causal conclusion can be made from the results.

4.1 Implications for Collaborative Tools Design

The results reported in this paper provide 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. In needed, managers or other stakeholders can take 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. [28], highlighting the difficulty of managing cross-boundary interdependencies, provide examples of interdependent teams that did not interact. The lack of appropriate communication resulted in difficulties at the time of integrating the various subsystems. Project management tools could use congruence information to provide a signaling mechanism that alerts project managers of potential issues with certain tasks prior to important milestones such as system integration.

The CSCW literature has proposed numerous tools to improve collaboration and task awareness among members of virtual teams. Some of the research efforts have focused on improving basic tools like email to better integrate the exchange of information with the tasks performed by individuals [2, 13]. The technique presented in this paper provides a way of identifying the email exchanges that are more relevant given the task interdependencies among individuals. This information would enable tools to provide an enhanced task management experience by, for instance, prioritizing to-do-lists and generating reminders to respond to task-specific emails based on the coordination requirements. We can think of this email sorting approach as a task-specific alternative to other social-based sorting techniques such as the one proposed by Fisher et al [11]. Web based collaborative tools like TeamSCOPE [18] could be also be enhanced to better present and distribute information based on the interdependencies that exist among the individuals' tasks.

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 straightforward 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 [27] and Palantir [26] have been proposed. This set of applications provides visualization and awareness mechanisms to aid developers to 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. ACKNOWLEDGMENTS

The authors would like to thank Matthew Bass, Thomas LaToza, Larry Macherrone, Jeff Roberts, Scott McCrickard, and the participants of the HCIC 2006 workshop for their insightful comments on previous versions of this work.

We gratefully acknowledge support by the National Science Foundation under Grants No. IIS-0414698, IIS-0534656 and IGERT 9972762, and by the Software Industry Center at Carnegie Mellon University and its sponsors, especially the Alfred P. Sloan Foundation.

6. REFERENCES

[1] Allen, T.J. *Managing the Flow of Technology*. MIT Press, Cambridge, MA, 1977.

[2] Belotti, V. et al. (2003). Taking email to task: the design and evaluation of a task management centered email tool. *In Proceedings of the Conference on Human Factors in Computer Systems* (CHI '03), Ft. Lauderdale, Florida, 2003, 345-352.

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

[4] Burton, R.M. and Obel, B. *Strategic Organizational Diagnosis and Design*. Kluwer Academic Publishers, Norwell, MA, 1998.

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

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

[7] Curtis, B. et al. Software psychology: the need for interdisciplinary program. *In Proceedings of the IEEE*, 74, 8 (Aug. 1986), 1092-1106.

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

[9] Daft, R.L. and Weick, K.E. Towards a model of organizations as interpretation systems. *Academy of Management Review*, 9, 2 (Apr. 1984), 284-295.

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

[11] Fisher, D., Hogan, B., Brush, A.J., Smith, M, Jacobs, A. Using social sorting to enhance email management. *Human-Computer Interaction Consortium* (HCIC '06), Fraser, Colorado, 2006.

[12] Galbraith, J. *Designing Complex Organizations*. Addison Wesley, Reading, MA, 1973.

[13] Gruen, D. et al. (2004). Lessons from the ReMail prototype. *In Proceedings of the Conference on Human Factors in Computer Systems* (CHI '04), Viena, Austria, 2004, 152-161.

[14] Handel, M. & Herbsleb, J.D. What is Chat Doing in the Workplace? *In Proceedings of the Conference on Computer Supported Cooperative Work* (CSCW'02), New Orleans, LA, 2002, 1-10.

[15] Henderson, R.M. and Clark, K.B. Architectural innovation: the reconfiguration of existing product technologies and the failure of established firms. *Administrative Science Quarterly*, 35, 1 (Mar. 1990), 9-30.

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

[17] Isaacs, E., Walendowski, A. and Ranganathan, D. (2002) Hubbub: A Sound-Enhanced Mobile Instant Messenger that Supports Awareness and Opportunistic Interactions. *In Proceedings of the Conference on Human Factors in Computer Systems* (CHI'02), Minneapolis, Minnesota, 2002, 179-188.

[18] Jang, C.Y., Steinfield, C. and Pfaff, B. (2000). Supporting awareness in a web-based collaborative system: The TeamSCOPE System. *In Proceedings of the Workshop on Awareness and the WWW, Conference on Computer Supported Cooperative Work* (CSCW '00), Pittsburgh, Pennsylvania, 2000.

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

[20] Levchuk, G.M. et al. Normative Design of Project-Based Organizations – Part III: Modeling Congruent, Robust and Adaptive Organizations. *IEEE Trans. on Systems, Man & Cybernetics*, 34, 3 (May. 2004), 337-350

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

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

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

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

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

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

[27] 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* (ECSCW '03), Helsinki, Finland, 79-98.

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

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

[30] Von Hippel, E. Task Partitioning: an innovation process variable. *Research Policy*, 19, 5 (Oct. 1990), 407-418