Communication Networks in Geographically Distributed Software Development

Marcelo Cataldo
Research and Technology Center
Bosch Corporate Research
Pittsburgh, PA 15212, USA
marcelo.cataldo@us.bosch.com

James D. Herbsleb
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213, USA
jdh@cs.cmu.edu

ABSTRACT
In this paper, we seek to shed light on how communication networks in geographically distributed projects evolve in order to address the limits of the modular design strategy. We collected data from a geographically distributed software development project covering 39 months of activity. Our analysis showed that over time a group of developers emerge as the liaisons between formal teams and geographical locations. In addition to handling the communication and coordination load across teams and locations, those engineers contributed the most to the development effort.

Author Keywords
Coordination, Geographically Distributed Product Development, Social Network Analysis, Collaboration Tools.

ACM Classification Keywords
D.2.9 [**Software Engineering**]: Management – *productivity, programming teams*. H.5.3 [**Information Interfaces and Presentation**]: Groups and Organization Interfaces – *computer-supported cooperative work, organizational design*.

INTRODUCTION

Over the past couple of decades, geographically distributed work has become pervasive and product development organizations are no exception. Factors such as access to talent, acquisitions and the need to reduce the time-to-market of new products are the driving forces for the increasing number of global product development projects [24, 27]. Unfortunately, this new trend has its costs. It is well established that physical proximity facilitates interactions among individuals working in R&D organizations [e.g. 2, 21]. Distance leads to numerous problems in communication and coordination, and ultimately, impacts the performance of product development teams [6, 21, 27]. A reduction in communication has been linked to failure to identify dependencies among work teams resulting in coordination problems [14, 18, 21, 38].

CSCW'08, November 8–12, 2008, San Diego, California, USA.

In order to support distributed teams, it is important to understand how information flows among teams and across sites, and the characteristics of the individuals that occupy key roles in the communication network. In this paper, we present a longitudinal examination of communication patterns among geographically distributed software development teams operating within an organization explicitly designed to allow teams to function as independently as possible. The data were collected from the two main communication tools used by the engineering organization and covered 39 months of development activity.

Our analysis showed several important findings: Over time, a *core group of developers emerged as the liaisons* between formal teams in different geographical locations. This replicates findings such as Allen's work on gatekeepers [2] that shows particular individuals play a key role in communication networks in engineering organizations. However, our results also show that those *individuals in the core not only perform a critical communication role but also they are the top contributors* to the actual development effort. That is, all the top performers in the project were part of the core group of engineers but they also continue to show high productivity in technical tasks. Moreover, the *communication core included other developers who rotated in and out of the core*, based on the dependencies of the technical work they were undertaking at the time. Finally, while the core developers participated in a disproportionate share of the communication overall, *they were even more disproportionately involved in cross-site* communication than in same-site communication.

THE LIMITS OF MODULARITY

March and Simon [26] argued that tasks should be divided into nearly independent parts and when interdependence is unavoidable, appropriate coordination mechanisms should be put in place. In the context of product development organizations, there is a close relationship between dividing development tasks into nearly independent parts and partitioning the system to be developed into nearly independent parts. Modularization is the approach typically used to minimize technical dependencies among the parts of a system

[10, 16, 37]. Baldwin and Clark [4, page 90] argued that modularization makes complexity manageable, enables parallel work and tolerates uncertainty. Moreover, Baldwin and Clark [4, page 89] argued that a modular *design structure* leads to an equivalent modular *task structure*, where one or more modules can be assigned to one organizational unit and work can be conducted almost independently of others [10].

In the context of software development, this approach was first articulated by Parnas [31] as modular software design. Parnas [31] argued that modules ought to be considered "work items" rather than "subprograms," emphasizing their role in allowing development work to continue independently and in parallel in the different modules. Modularization is thought to create loose coupling between tasks, which is generally considered to be appropriate when teams are geographically distributed [30].

The modularization argument relies on the assumption that a simple and obvious relationship exists between the structure of the product and the structure of the tasks required to develop the product. Hence, by reducing the technical interdependencies among modules, the modularization theories argue, task interdependencies are reduced, thereby reducing the need for communication among work groups.

While the modularity approach to coordination has proven very useful, it is becoming clear that it is by itself insufficient to resolve coordination problems in product development. First, there is recent empirical evidence indicating that the relationship between product structure and task structure is not as simple as previously assumed, and the theorized similarity between those structures diminishes over time [8]. One important factor is the dynamic nature of task dependencies in software development. The requirements of a software system change as time progresses and, in many instances, they only become known over time [25]. Uncertainty and change in requirements can result in minor alterations or cause significant modifications such as adding or eliminating a feature of the product. These events, typically, require a well coordinated effort that might affect tens or hundreds of modules.

Second, reducing communication between teams responsible for modules can lead to difficulties, since, like most subsystems [33], modules are rarely fully independent. The product development literature argues that information hiding, a primary strategy to achieve modularity in software development, leads to minimal communication between teams. This in turn causes variability in the evolution of different components resulting in integration problems [38]. In the context of software development, de Souza and colleagues [14] found that information hiding led development teams to be unaware of others teams' work, resulting in coordination problems. Grinter and colleagues [18] reported similar findings for geographically distributed software development projects. The authors highlighted that the main consequence of the reduced need of communication between teams was increased costs because problems were discovered too late in the development process.

Finally, the act of developing a software system consists of a collection of design decisions that often introduce design or implementation constraints that change the dependencies among the various parts of the system. These changes can generate new coordination requirements that are quite difficult to identify *a priori*, particularly when they affect the product architecture [20, 35]. Failure to discover the changes in coordination needs can have a profound impact on the quality of the product [13] and on productivity [21].

"GATEKEEPER" COMMUNICATION NETWORKS

The limitations of the modular design strategy suggest that communication among teams will be essential in order to coordinate project work. Organizational and geographic barriers to communication can be overcome by individuals in key roles who facilitate and promote the interaction between teams [2, 3, 19]. Several definitions of those key positions have been proposed in the product development literature [19]. Examples are "alliance champion", "external liaison", "gatekeeper", and "process promoter". Although those definitions differ slightly from each other in their theoretical underpinnings, the overarching theme is that those individuals perform a different type of activity than the rest of the members of a R&D group and their task is critical for the success of a project. Those key people have access to different sources of information and they are capable of synthesizing the information in a way useful for the various groups so they can to better perform their development activities [19].

The use of "liaison" or "gatekeepers" to manage the dependencies between teams has also been proposed as a mechanism for facilitating coordination in geographically distributed software development [32]. As engineers perform their development tasks, critical information and knowledge about the parts of the system involved in the tasks at hand is exchanged. As software development tasks change over time, developers get the opportunity to gain access to new information and knowledge about the technical properties of different parts of the system. This system of social relationships, which we will refer to as a *communication network*, is an evolving entity. If gatekeepers are strategically embedded in the communication networks, they can acquire the necessary knowledge to discover the relevant technical and task dependencies.

This paper attempts to shed light on how communication networks evolve in geographically distributed projects in order to understand how organizations overcome the limits of the design modularity strategy. In particular, we examine the following research questions.

Research Questions

Past research on communication patterns in R&D organizations suggested the existence of "gatekeeper networks". Using the graph theoretic concept of strong components,

Allen [2] showed that gatekeepers tended to be part of a highly clustered group of individuals bridging communication between disconnected members of the R&D laboratories. Allen's findings lead to several research questions:

RQ1: Does a highly interconnected group of people take on a disproportionate share of overall communication?

RQ2: Does a highly interconnected group of people take on a disproportionate share of cross-site communication?

Gatekeepers in R&D organizations are also perceived as very technically competent individuals who are able to interpret several sources of information, translate them and synthesize them to be consumed by development teams [2, 19]. We are interested in understanding the characteristics of people who assume a gatekeeper role in software development organizations. It has long been known that gatekeepers are technically capable, but does the gatekeeper role force them to focus primarily on communication, or do they also remain highly productive in technical tasks? Are there other characteristics that also bring individuals into the core of the communication network?

RQ3: Are the most productive technical people part of the core of the communication networks?

RQ4: What other characteristics can lead to a technical person becoming part of core the communication networks?

DESCRIPTION OF THE RESEARCH SETTING

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 about thirty-nine months of development activity and the first four releases of the product. During that period of time the company had a maximum of one hundred and nineteen developers. However, five of them were part of the company for a period of less than 6 months so we did not consider them in our analysis. The one hundred and fourteen developers that were included in our analysis were already part of the development organization prior to the time period covered in our analysis or joined the company in the first four months of the examined time period. The developers were organized in eight teams and distributed in three research and development locations. Each team had a manager but the developers did not have formal roles such as team lead or liaison. All engineers were encouraged to seek interaction with members of other teams if they considered it necessary. All the developers worked full time on the project during the time period covered by our data.

Because of the way work was allocated to teams, this project made a near-ideal setting for investigating the limits of modularity. The organization had been put together to build a single product, and this was the only product developed during the period of our study. The product was designed to be highly modular, and the organization was designed to fit the structure of the product. Each component was assigned to a single team, whose members were collocated. Because of this match between teams and components, virtually all of the inter-team coordination represents issues that the modular approach did not resolve.

The development tasks were identified by modification requests (MRs) which represent defects in the software or functionality enhancement requests. Software developers communicated and coordinated their development tasks in several ways. Opportunities for interaction exist when working in the same formal team or when working in the same location. For instance, all the development teams had periodic meetings as frequent as once or more times a week. Developers also used a range of communication tools to interact and coordinate their work such as email, an online-chat system (Internet Relay Chat - IRC), video conference, and a development task tracking system. One of the authors interviewed several developers, who identified the online-chat system (IRC) as the primary communication means for development and debugging work. The second most commonly used tool was the MR tracking system which not only tracked requests as they were opened, assigned, and resolved, but provided text chat capability for each request. In addition, developers indicated that they used email and video-conferences, but primarily for design and architectural definition activities. Given those patterns of communication means usage, we collected communication and coordination information from the online chat and the MR-tracking system. In the rest of the document, we will refer as IRC communication or coordination as such activities carried out using the online-chat system. We will refer to as MR communication or coordination when those activities took place using the MR-tracking system.

Data Collection and Measures

On a daily basis, developers interacted with other engineers in the same or other laboratories using IRC. The company established several channels based on formal teams as well as special projects. For example, team A is responsible for components 1 and 2 and there is a channel A in IRC. Then, any engineer that requires information about components 1 and 2 would typically communicate with other engineers in channel A. In order to preserve the valuable technical information discussed on IRC, the company logged the channels associated with formal teams and special projects. This repository provided the historical data that allowed us to reconstruct the patterns of interaction and coordination amongst the developers. In order to identify the relevant interactions, we used the modification requests as guides.

As part of a larger research project, three raters, blind to the research questions, examined the IRC logs corresponding to all the recorded channels and associated with the modification requests in our dataset. This data coding effort took approximately 14 months. We considered the alternative of using a tool, such as PieSpy [29], to extract relational data from IRC. Unfortunately the tool was not able to accurately

identify which modification request was discussed in a given interaction. Since our unit of analysis is the modification request, it was critical that we be able to correctly associate specific communications with the modification requests to which they referred, and to eliminate communications that did not concern modification requests.

Since the work on a MR could extend over days, weeks or even months, the raters were instructed to examine IRC logs through out the entire period of time associated with each MR. When interacting, developers could refer to the MR id number (e.g. "<developer01> developer02: have you looked at bug 12345") or to the problem the MR represents without any explicit reference to the MR (e.g. "<developer01> does anyone know why would RPC call 123 returns the error code 12345?"). The raters were given a description of the problem associated with each modification request in order to be able to identify the latter type of interactions. The outcome of the data coding process was a list of time stamped events indicating <developer A> interacted with <developer B>. We assessed the reliability of the raters' work by having them code 10% of the MRs by all three raters. Comparisons of the obtained networks showed that 98.2% of the networks had the same set of nodes and edges.

From the list of interactions associated with each modification request, we constructed the communication networks on a monthly basis grouping into a single monthly network all the interactions among developers from each MR that occur in each particular month. The nodes in the networks represent the one hundred and fourteen developers in the company's engineering organization. The networks do not consider managers or other higher level decision making individuals. If any of the developers did not participate in any discussion on the IRC logs for a particular month, he or she would be represented in the network as a node without connections, in other words, an isolate.

The company also used a MR tracking system to monitor the progress of development tasks and to facilitate the exchange of information and discussion about the development tasks. For example, as defects are debugged, developers post information regarding their findings and might request information from other developers that would provide useful feedback. We defined an interaction between developers *i* and *j* only when both *i* and *j* explicitly commented in the MR report. We focused on the developers that explicitly commented on the MR report because the MR tracking system sent email to all the addresses in a CC list every time an MR is updated. Therefore the number of recipients of updates could be significantly larger than the set of people actually discussing the MR. We also ignored comments automatically generated by the workflow tool (e.g. changes to the status of the task). Then we used these exchanges of information to construct communication networks amongst developers. In this case, the data collection process was automated by using a script that interacted with the modification request tracking system and constructed the monthly social networks.

We collected several individual-level measures about the developers. Using the relational data from the MR system and IRC, we computed two network measures that have been shown to relate to individual-level performance: the degree centrality [17] and the network constraint [7]. We also evaluated several other network measures, such as betweenness centrality and eigenvector centrality, but they were highly correlated with degree centrality. Demographic data about the developers such as their programming and domain experience and level of formal education were collected from archival information provided by the company's human resources department.

Measuring individual-level performance in software development is not a trivial task. The concept of performance could be interpreted across different dimensions such as the amount of code produced, the quality of that produced code in terms of lack of defects, efficiency and maintainability. Previous research has proposed different approaches to measure individual-level software development performance and, in this study, we focused on measures based on the amount of code created (see, e.g., Curtis [12]). We used two measures of performance. First, a measure of contribution to the development effort was defined in terms of the number of changes, instead of a more traditional lines-of-code measure, which allows us to control for variability in developers' coding style (e.g. developers who might have a more verbose versus a more compact coding style). Moreover, the development organization studied encouraged developers to submit changes to the version control system that constituted logical pieces of work as a single commit. Hence, the measure *Number of Changes Contributed* represents an appropriate measure of task performance. A second measure of performance is represented by the number of modification requests resolved by each developer, *Number of MRs Resolved*. Both measures were highly correlated (0.68, $p < 0.01$) because all changes to the source code were represented by a modification request.

RESULTS

We organized the presentation of our results around our research questions.

RQ1: Does a highly interconnected group of people take on a disproportionate share of overall communication?

Using the interaction data from IRC and the MR-tracking system, we constructed monthly communication networks. Figure 1 shows months 10, 20 and 30 from the IRC data. The general pattern of the communication networks is a core-periphery structure [5] suggesting that a particular group of developers are at the center of the coordination activities and the exchange of information among engineers. The core-periphery patterns were analytically confirmed by using Borgatti and Everett's [5] method for fitting network patterns to an idealized core-periphery structure. We used Borgatti and Everett's continuous model that

gives a fit value between 0 and 1. The closer the fit is to 1, the more similar a particular network is to a core-periphery structure. The average fit across all 39 months was 0.721 with a minimum fit of 0.568 and a maximum one of 0.858. The rest of the developers seem to rely solely on interactions with the centrally positioned developers for coordinating their tasks. Our coordination data from the MR-tracking system showed the same core-periphery pattern.

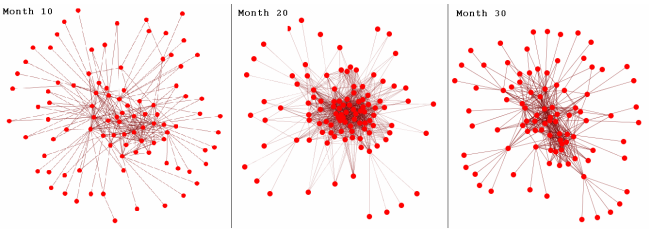

Figure 1: Over Time Coordination Patterns using MR system

In other words, the strong core-periphery patterns suggest the communication network features a highly interconnected and relatively small group (averaging about 26% of the total) of people who play a special role as communication hubs.

RQ2: Does a highly interconnected group of people take on a disproportionate share of cross-site communication?

The role of the core group in terms of communication across geographical locations was statistically examined using an ANOVA analysis to evaluate the frequency of interaction in a 2 x 3 factorial design where dyads were classified along two dimensions: same geographical location (yes or no) and position in the coordination network (both nodes in the core, both nodes in the periphery or a node from each group). We used the MR and IRC coordination data aggregated at the level of product release and the core groups on a monthly basis were identified using Borgatti & Everett's [4] method. Since the observations (the dyads) are not independent, we assessed the ANOVA results using a random replication procedure. We used 1000 and 5000 replications and all ANOVA results were consistent.

As expected, we found a statistically significant main effect of geographical location (F=74.70, p<0.001) with much more frequent communication within a site than across sites. We also observed the obvious main effect for position in the network (F=93.95, p<0.001), which was inevitable given that frequent communicators end up in the core.

We answered our research question RQ2 by examining the interaction term, finding a significant effect on the frequency of communication (F=15.51, p<0.001) (see figure 2). *Developers in the core handle even a larger proportion of the cross-site communication than they do communication within a site.*

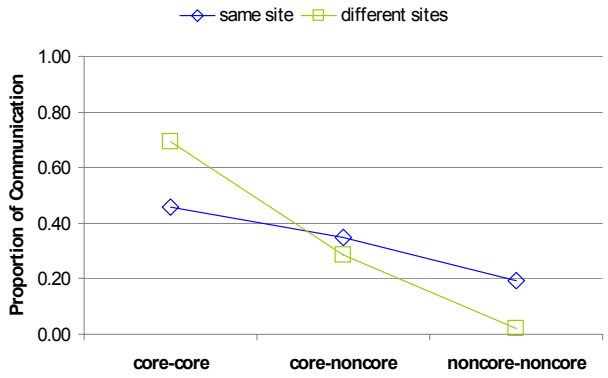

Figure 2: Proportion of Communication by Network Position and Location

For illustrative purposes, figure 3 shows the communication network from month 17 from the IRC coordination data where the developers are color-coded based on geographical location. Consistent with the results from the ANOVA analysis, we observe that a group of developers in the core of the network act as conduits to other geographical locations for the developers in the periphery. The same core-periphery pattern emerged in the case of communication across formal teams where the core was composed of individuals from all eight formal teams. The existence of these gatekeepers replicates the findings Allen [2] encountered in R&D organizations, and extends them to geographically distributed teams.

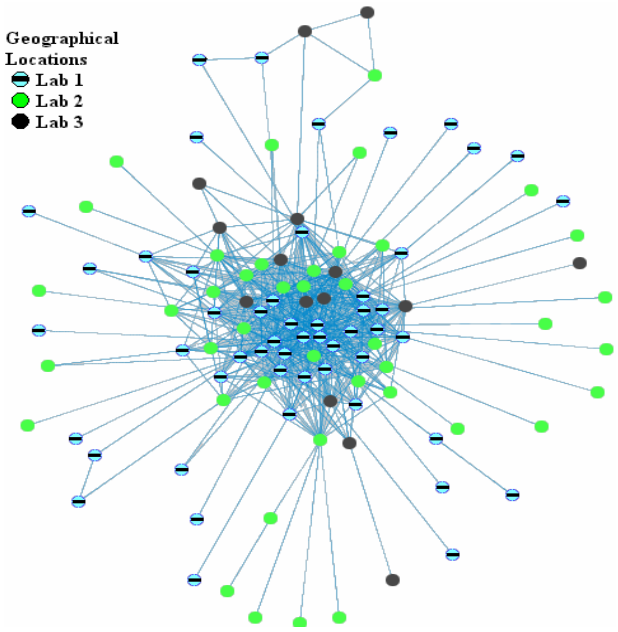

Figure 3: Communication Patterns across Locations (from IRC data)

RQ3: Are the most productive technical people part of the core of the communication networks?

We examined this question both qualitatively and quantitatively. The qualitative analysis suggested the highest-performing developers tended be in the core. However,

productivity is the outcome of a number of different factors. Therefore, we also used a multi-level regression model to examine if the developers' degree centrality was associated with productivity.

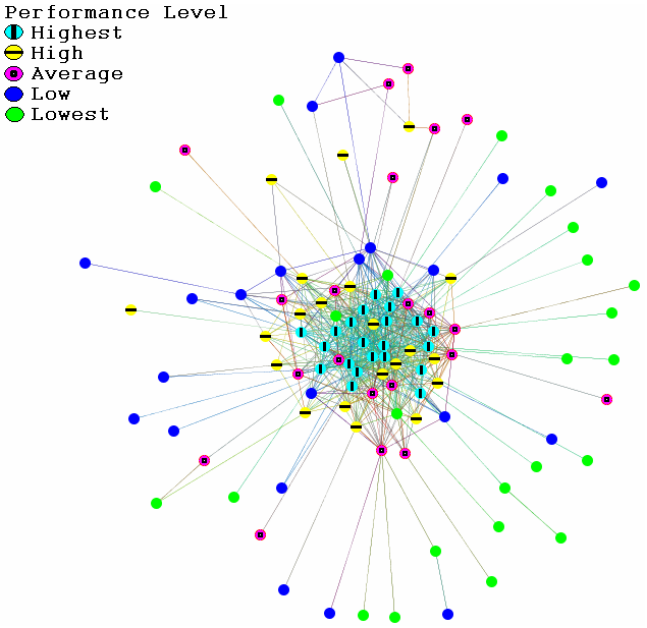

Figure 4: Coordination Patterns and Productivity (from IRC data)

Qualitative Analysis

In order to gain a better understanding of the composition of the core group in the communication networks, we relate membership to the core group to the developers' contribution to the development effort. Past research in software engineering has found that a large portion of the modifications to a software system is typically done by a small number of developers [28]. In our dataset, we encountered that 18 to 20 developers contributed approximately 50% of the code in the software system. That group of developers corresponded to about only 18% of the engineers in the project. Given this pattern, we ranked the developers in terms of the amount of code contributed to the development effort on a monthly basis and we divided the ranking into five groups with each group corresponding to 20% of the developers. Figure 4 shows an example of a communication network (month 17) where each developer is categorized into a productivity group. We observe that the majority of the developers in the core are the highest performers while less performing developers tend to remain in the periphery of the communication network. However there are several interesting cases. There are several high performing engineers that are in the periphery and they seem to coordinate their work minimally. On the other hand, there are low performing individuals positioned very centrally in the communication network.

We performed additional analysis to examine whether the patterns shown in figure 4 persisted over time. First, we compared the monthly communication networks along two dimensions: how many developers were in the core of the communication network in each month, and how many top performing developers were part of the core in each month. Again, we identified the core groups on a monthly basis using Borgatti & Everett's [5] method. As depicted in figure 5, the core group averaged 30, with a minimum number of 14 and maximum of 42 engineers. In addition, the number of engineers from the highest productivity group that were part of the core group in the communication networks ranged from 9 to 22 over the 39 months of data.

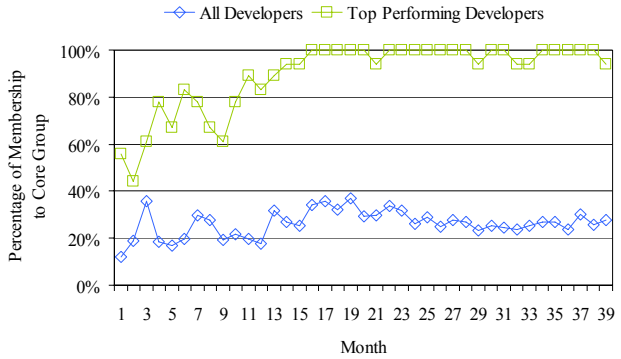

Figure 5: The Size of the Core Group over Time and Top Performers Membership

We also observe in figure 5 that during the first 1/3 of the time covered by our analysis, the composition of the core group varied significantly. After month 15, we see that most of the top performers consistently belong to the core on the communication network. However, there are several instances where low productivity developers are also part of the core coordination group (see Figure 6). We examined those cases in detail when we address research question 4.

Quantitative Analysis

Our longitudinal dataset had characteristics that render traditional linear regression models inadequate for statistical analysis. As is the case with any longitudinal dataset, the autocorrelation between the observations of the same measures over time will violate the independence assumptions of a traditional linear model. In order to correctly deal with the lack of independence stemming from the longitudinal nature of the dataset, we used a multi-level model [34]. The multi-level modeling approach allows variation at several levels within the model. The specification of a multi-level model includes fixed effects and random effects that may be applied to multiple variables for a given stream of longitudinal data.

In our analysis, we have a stream of data for each developer and the model allows variation of both the intercept and the influence of time and formal team membership on the productivity of individuals. Our models included independent factors that the software engineering literature has found to be important predictors of productivity such as individual level attributes of the engineers, characteristics of the development tasks and the two network measures: degree

centrality and network constraint. As described earlier, we used two different individual-level productivity measures. Overall, the pair-wise correlations had acceptably low levels (below 0.20) with the exception of the correlation between network centrality and network constraint which was 0.421 for the IRC data and 0.274 for the MR data.

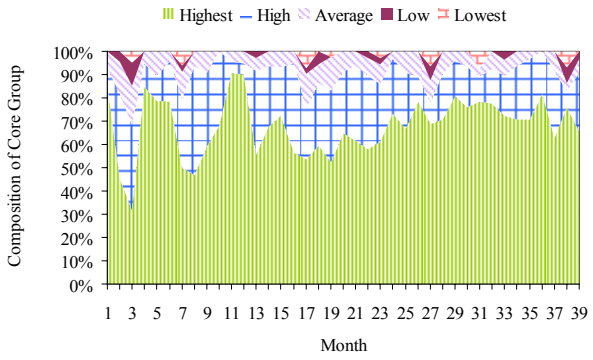

Figure 6: Composition of the Core Group over Time by Productivity Levels

In table 1, we report the effects of the various factors on the *Number of Changes Contributed* productivity measure using the IRC (models I & II) and MR (models III & IV) coordination data to compute the network centrality and network constraint measures. Table 2 shows the coefficients of the multi-level regression using the *Number of MRs Resolved* as individual-level performance measure.

Communication over IRC		
	Model I	Model II
Intercept	2.621**	2.715**
Time	0.008**	0.007**
Education (log)	-0.031	-0.027
Domain Exp. (log)	0.079**	0.076**
Avg. Change Size (log)	-0.474**	-0.434**
Degree Centrality	2.621**	2.818**
Network Constraint		0.317**
AIC	7010	6931
Deviance Explained	51.03%	51.60%
Communication over MR-Tracking System		
	Model III	Model IV
Intercept	3.569**	3.576**
Time	0.003**	0.003**
Education (log)	-0.025	-0.025
Domain Exp. (log)	0.051+	0.048+
Avg. Change Size (log)	-0.532**	-0.531**
Degree Centrality	1.378**	1.389**
Network Constraint		0.031
AIC	7485	7491
Deviance Explained	47.71%	47.68%

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

Table 1: Results of the Multi-level Regression for the effects on Number of Changes Contributed

The results from table 1 suggest that over time, developers become more productive in terms of changes submitted as the statistically significant and positive coefficient on the variable *Time* indicates. Consistent with prior research [12, 13], higher levels of *Domain Experience* increased the number of changes submitted by the developers on a monthly basis. We also examined the impact of programming experience but the measure was highly correlated to domain experience and had similar impact to the one reported for domain experience. As expected, the larger the average size of the modifications made by the developers, the lower the number of changes those developers contribute to the development effort as indicated by the negative coefficient of the *Avg. Change Size* measure.

In terms of the network measures, our results are consistent with past research showing that individuals centrally positioned in the communication network were more likely to exhibit higher levels of performance [36]. However, we also found that those individuals that were embedded in a highly interconnected ego network (*Network Constraint*) were even more productive, suggesting the relevance of belonging to the core group in the communication network. Although our analysis is at the individual-level, it is important to relate these findings to past research showing that geographically distributed groups or organizations with hierarchical social networks tended to coordinate better [22] and performed better [11] than those with core-periphery or flatter structure. An important factor that may be at play here is the dynamic nature of work dependencies that emerges in software development and potentially in other knowledge-intensive activities. Such an environment could reduce significantly the routineness of the tasks and increase uncertainty around the tasks, leading to non-hierarchical patterns of communication. In fact, Ahuja and Carley [1] showed that hierarchical social networks developed in virtual organizations when task exhibited higher levels of routineness.

The results reported in table 2 were consistent with those reported in table 1 with the exception of the impact of *Time* which has a negative impact on productivity. This finding suggest that over time, engineers tend to resolve less modification requests, possibly because the MRs require more development effort or they represent more complex tasks.

In sum, the results from the qualitative and quantitative analysis suggest that membership in the core is strongly associated with productivity level of the developer. In an effort to explore the nature of the relationship between productivity and degree centrality and network constraint, we investigated the personal characteristics of developers and how they varied across levels of productivity. Our thought was that there are numerous factors that could explain this relationship. For instance, the software engineering literature suggests that top developers typically have an order of magnitude better performance than average developers [12]. Such superior skill might make those developers the key source of technical knowledge and, consequently, they

become involved in the communication and coordination structures of the project. We compared the productivity groups across several individual-level factors: programming and domain experience, education level, and tenure in the company. Our analysis did not find any statistically significance differences across the groups.

Communication over IRC		
	Model I	Model II
Intercept	0.984**	1.020**
Time	-0.011**	-0.005**
Education (log)	-0.027	-0.020
Domain Exp. (log)	0.190*	0.296*
Avg. Change Size (log)	-0.102**	-0.129**
Degree Centrality	4.246**	3.949**
Network Constraint		0.891**
AIC	8460	8141
Deviance Explained	24.27%	27.16%
Communication over MR-Tracking System		
	Model III	Model IV
Intercept	1.744**	1.806**
Time	-0.007**	-0.007**
Education (log)	-0.054	-0.055
Domain Exp. (log)	0.163**	0.164**
Avg. Change Size (log)	-0.105**	-0.113**
Degree Centrality	2.939**	2.900**
Network Constraint		0.623^{+}
AIC	9204	9172
Deviance Explained	17.44%	17.76%

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

Table 2: Results of the Multi-level Regression for the effects on Modification Requests Resolved

RQ4: What other characteristics can lead to a technical person becoming part of the core of communication networks?

We also investigated other factors that may cause developers to enter the core. Our qualitative examination of the data (see figures 4 through 6) indicated that some developers who were not high performers occasionally entered the core group, at least temporarily. In order to investigate this aspect of the core group composition in more detail, we examined the nature of the development tasks across the productivity groups. Specifically, we looked at differences in the average number of files affected by a modification request. The thought was that task dependencies may force developers into the center of the communication network, independently of their productivity, because of the need to interact and coordinate with individuals from different teams that had the relevant knowledge of different parts of the software system.

Our analysis revealed important differences across productivity groups in terms of the average number of source code files affected by modification requests. In fact, for several months in our data (e.g. 3, 8, 17, 27, and 38), we found that some developers in the lowest two productivity groups worked on modification requests that affected, on average, the highest number of source code files. This finding is clearly depicted in figure 6 which shows that in certain months (3, 8, 17, 27, and 38) the composition of the core group involves a higher number of lower productive engineers. Our examination of the modification requests and changes to the source code indicated that for those months, the low-performing developers worked on features of the system that cut across numerous subsystems such as tracing and security functionalities. Modifications in that type of functionalities require coordinating work across several groups of individuals. These data suggest that it is the dependencies in the technical work that appear to drive some of the lower performing developers temporarily to the core of the communication network. It is important to highlight that these findings refer to only five low-performing developers. The other low-performing engineers did not exhibit the same task and interactions patterns. One unexpected observation is that several lower-performing developers who worked on cross-cutting concerns tended to move up one or two categories in the productivity ranking after the months where they were part of the core group of the communication network. However, the improvement in productivity did not translate into a consistent membership in the core group of the communication networks over time.

DISCUSSION

In this paper, we have presented a longitudinal analysis of communication activities in a geographically distributed software development project and the results provide three main contributions to the CSCW literature. Prior work has shown that gatekeepers are technically competent and they are often first level managers in development organizations [e.g. 2]. Our analyses showed that the core people in the studied project are, by and large, not just technically competent but, in fact, they had major direct contributions to the development effort. In other words, the individuals in the core are also highly productive. This is a novel result because past research suggests that the communication hubs would evolve into communication specialists and, consequently, reduce their level of productivity.

Our findings also showed that the communication hubs play an even greater role in communicating across sites than within a single site. Giving that distributed product development organizations are becoming the norm, also it is important to further develop our understanding about how information flows through distributed projects. Our results are a step forward in that direction.

Finally, our study showed that the core group consisted of two kinds of individuals: not only gatekeepers, also individuals drawn at least temporarily into the hub by their work assignments, which is the new result. This suggests that communication tools must retain sufficient role flexibility to permit this movement, rather than locking people into core or non-core roles.

Limitations of the study

While the development organization we studied was a good choice for research seeking to explore the limits of modularity as a tool for coordination, these very same characteristics – new development effort, single product, organization designed around modular product structure – make it somewhat unusual. In organizations with multiple products and less precise mapping of modules onto teams, the effects observed here may be diminished by other drivers of communication and performance. It is also not clear how far beyond software development work these results can be generalized. While it seems reasonable to speculate that product structure will profoundly influence coordination in any product development organization [20, 35], software development may have unique characteristics.

It is also worth pointing out that we did not have the opportunity to observe all communication, for example face-to-face, telephone, and video conference. While our interviews gave us some confidence that much of the technical communication occurred in the channels we observed, the patterns of interactions across those channels we did not have access to may have looked different. The effects of centrality that we observed, for example, may only apply to centrality in networks over particular kinds of textual media. Finally, we recognize that our productivity measures have limitations but it is important to notice that the measures used in this study are well known and commonly used in software engineering research.

Implications for Tool Design

The analysis and results reported in this study suggests several research directions in relation to tools to support distributed product development organizations. First, tools could focus their attention on a project-level view of coordination, leveraging data stored in software development repositories (e.g. MR tracking systems) to assess the project-wide coordination patterns of geographically software development teams. Several tools that depict information regarding the structure of communication and coordination patterns in relation to work dependencies have recently been proposed [e.g. 15]. However, our study show those tools could be extended by combining communication patterns data with information about contributions or progress of tasks which would allow specific stakeholders, such as project managers or team leads, to identifying potential problem areas and take appropriate corrective measures, if necessary.

Secondly, communicators in the core remained highly productive, this suggest that even for hub communicators, tools must support easy movement between technical work and communication. Finally, it perhaps also calls into question the degree of difficulty that interruptions cause – it has not prevented the most frequent communicators from being highly productive. While others have argued that interruptions are important for managers [23], our results might suggest that the information gained from interruptions may be undervalued. Alternatively, it may suggest that some people are more able to swap context than others. These topics represent important issues for future research in the area of tool design for supporting distributed development organizations.

Implications for Future Research

The finding that core developers play a particularly large role in facilitating communication across sites may represent an effective adaptation or it may mean that the communication core is likely to become a bottleneck. In this project, where conditions allowed for an organizational design specifically adapted to the product structure, we would expect that cross-site coordination requirements would be relatively low compared to many projects. With much larger volumes of communication to deal with, it is not clear that core members can continue to act as cross-site hubs while also maintaining their very high productivity. Future research on other organizational arrangements should be able to shed light on this.

More research is required to better understand the relationship between development tasks that promote interactions among engineers and the potential gains in development productivity. In addition, future research should examine if tasks such as the implementation of cross-cutting concerns, are mechanisms that could promote the development of communication and coordination conduits among formal teams and development locations.

In our research setting, the "liaisons" emerged over time from each development group, contrary to view typically discussed in the literature where these key roles are formally established [3, 19, 32]. In fact, exploratory analyses of communication patterns in distributed software development organizations suggest that when "liaisons" are formally defined minimal communication channels emerge between formal teams and geographic locations [9]. Individuals in such formal roles, with different expectations and responsibilities from the rest of the engineers in a software development effort, might face important challenges stemming from the dynamic nature of technical and task dependencies. Future research should examine the differential impact, if any, of formal versus emergent "liaisons" roles.

ACKNOWLEDGEMENTS

We gratefully acknowledge support by the National Science Foundation under Grants IIS-0414698 and IIS-0534656, the Software Industry Center at Carnegie Mellon University and its sponsors, especially the Alfred P. Sloan Foundation.

REFERENCES

1. Ahuja, M.K. and Carley, K.M Network Structure in Virtual Organizations. *Org. Science*, 10, 6 (1999), 741-757
2. Allen, T.J. *Managing the Flow of Technology*. MIT Press (1977).

3. Ancona, D.J. and Caldwell, D.F. Bridging the boundary: external activity and performance in organizational teams. *Administrative Science Quarterly*, 37 (1992).
4. Baldwin, C.Y. and Clark, K.B. *Design Rules: The Power of Modularity*. MIT Press (2000).
5. Borgatti, S.P. and Everett, M.G. Models of Core Periphery Structures. *Social Networks*, 21 (1999), 375-395.
6. Brown, S.L. and Eisenhardt, K.M. Product Development: Past Research, Present Findings, and Future Directions. *Academy of Management Review*, 20, 2 (1995).
7. Burt, R.S. Structural Holes: T*he Social Structure of Competition*. Harvard University Press, 1992.
8. Cataldo, M. et al. Identification of Coordination Requirements: Implications for Design of Collaboration and Awareness Tools. *Proc. CSCW'06*, ACM Press (2006).
9. Cataldo, M. and Herbsleb, J.D. Communication patterns in geographically distributed software development and engineers' contributions to the development effort. *Proc. CHASE'08*, ACM Press (2008).
10. Conway, M.E. How do committees invent? *Datamation*, 14, 5 (1968), 28-31.
11. Cummings, J.N and Cross, R. Structural Properties of Work Groups and their Consequences for Performance. *Social Networks*, 25 (2003), 197-210.
12. Curtis, B. *Human Factors in Software Development*. Ed. by Curtis, B., IEEE Computer Society, 1981.
13. Curtis, B., Kransner, H. and Iscoe, N. A field study of software design process for large systems. *Comm. of ACM*, 31, 11 (1988), 1268-1287.
14. de Souza, C. et al. How a Good Software Practice Thwarts Collaboration: The multiple roles of APIs in Software Development. *Proc. FSE'04*, ACM Press (2004).
15. de Souza, C. et al. Supporting Collaborative Software Development through Visualization of Socio Technical Dependencies. *Proc. GROUP'07*, ACM Press (2007).
16. Eppinger, S.D. et al. A Model-Based Method for Organizing Tasks in Product Development. *Research in Engineering Design*, 6 (1994), 1-13.
17. Freeman, L.C. Centrality in Social Networks: I. Conceptual Clarification. *Social Networks*, 1 (1979), 215-239.
18. Grinter, R.E., Herbsleb, J.D. and Perry, D.E. The Geography of Coordination Dealing with Distance in R&D Work. *Proc. GROUP'99*, ACM Press (1999).
19. Hauschildt, J. and Schewe, G. Gatekeeper and process promoter: key persons in agile and innovative organizations. *International Journal of Agile Management Systems*, 2 (2000), 96-103.
20. Henderson, R.M. and Clarck, K.B. Architectural Innovation: The Reconfiguration of Existing Product Technologies and the Failure of Established Firms. *Administrative Science Quarterly*, 35, 1 (1990), 9-30.
21. Herbsleb, J.D. and Mockus, A. An Empirical Study of Speed and Communication in Globally Distributed Software Development. *Trans. on Soft. Eng.*, 29, 6 (2003).
22. Hinds, P. and McGrath, C. Structures that Work: Social Structure, Work Structure, and Coordination Ease in Geographically Distributed Teams. *Proc. CSCW'06*, ACM Press (2006).
23. Hudson, J.M. et al. "I'd be overwhelmed, but it's just one more thing to do": availability and interruption in research management. *Proc. CHI'02*, ACM Press (2002).
24. Karolak, D.W. *Global Software Development: Managing Virtual Teams and Environments*, IEEE Computer Society (1998).
25. Leffingwell, D. and Widrig, D. *Managing Software Requirements: A Use Case Approach, 2nd Edition*. Addison-Wesley (2003).
26. March, J. and Simon, H.A. *Organizations*. Wiley, 1958.
27. McDonough, E.F., Kahn, K.B. and Barczak, G. An Investigation of Global, Virtual and Collocated New Product Development Teams. *Journal of Prod. Innovation Mgmt.*, 18 (2001), 110-120.
28. Mockus, A. and Weiss, D.M. Predicting risk of software changes. *Bell Labs Tech. Journal*, Apr 2000, 169-180
29. Mutton, P. Inferring and visualizing social networks on Internet Relay Chat. *In Proceedings of the Information Visualization Conference (IV '04)*, 2004.
30. Olson, G.M. and Olson, J.S. Distance Matters. *Human-Computer Interaction*, 15, 2 & 3 (2000), 139-178,
31. Parnas, D.L. On the criteria to be used in decomposing systems into modules. *Comm. of ACM*, 15, 12 (1972).
32. Sangwan, R. et al. *Global Software Development Handbook*, Auerbach Publishers (2006).
33. Simon, H.A. *The Sciences of the Artificial*, MIT Press (1996).
34. Singer, J.D. and Willet, J.B. *Applied Longitudinal Data Analysis*. Oxford University Press, 2003.
35. 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 (2004), 1674-1689.
36. Sparrowe, R.T. et al. Social networks and the performance of individuals and groups. *Academy of Management Journal*, 44, 2 (2001), 316-325.
37. Sullivan, K et al. The Structure and Value of Modularity in Software Design. *Proc. FSE'01*, ACM Press (2001).
38. Yassine, A. et al. Information Hiding in Product Development: The Design Churn Effect. *Research in Engineering Design*, 14 (2003), 145-161.