Coordination in Large-Scale Software Development: Helpful and Unhelpful Behaviors

Christopher Poile
Dept. of Management Sciences
University of Waterloo
Waterloo, ON, Canada
cpoile@uwaterloo.ca

Andrew Begel, Nachiappan Nagappan
Microsoft Research
Redmond, WA
{andrew.begel, nachin}@microsoft.com

Lucas Layman
Dept. of Computer Science
North Carolina State University
Raleigh, NC
lucas.layman@ncsu.edu

September 28, 2009

Abstract

Large-scale software development requires coordination within and between very large engineering teams which may be located in different buildings, on different company campuses, and in different time zones. At Microsoft Corporation, we studied a 3-year-old, 300-person software application team based in Redmond, WA to learn how they coordinate with three intra-organization, physically distributed dependencies: a platform library team also in Redmond; a team three time zones away in Boston, MA; and a team in Hyderabad, India. Thirty-one interviews with 26 team members revealed that coordination was most impacted by issues of communication, capacity and cooperation. Distributed teams faced additional challenges due to time zone and cultural differences between the team members. We support our findings with a survey of 775 engineers across Microsoft who described their experiences managing coordination in their own software products. We suggest new processes and tools to improve team coordination.

1 Introduction

Coordination between software development teams is one of the most difficult-to-improve aspects of software engineering. Kraut and Streeter argue that the software industry has been in crisis mode for its entire existence, and a root cause is the difficulty in coordinating work between teams of developers [27]. Researchers have studied professional software development teams empirically to gain greater understanding of how software development processes, tools, and people impact coordination [18, 40, 46, 35]. The importance of intra- and inter-team coordination is a foremost concern as software development increasingly becomes globally distributed, and remains a persistent challenge in other disciplines as well.

We view coordination as decision-making and action requiring *communication*, *capacity* and *cooperation*. These three components of coordination are necessary, but by themselves insufficient, for coordination to take place. Communication is necessary because person A needs to communicate to person B, in some form, what needs to be done, and B needs to understand the communication. Capacity is necessary because B needs to be *able* to do what is required of them. Cooperation is necessary because B needs to be *willing* to do what is required of them. If any of the three necessary components are lacking, the outcome will be less than ideal.

Viewing coordination through this framework leads us to ask several research questions:

1. What kinds of behaviors are associated with being helpful or unhelpful to others?

2. How do individuals on a software team communicate to get work done?
3. How do software teams manage dependencies on a personal level?

To understand inter- and intra-team dependencies in large scale software development, we conducted a large interview-based study of a 300-person Microsoft software development group, of which two of its teams are distributed globally. We collected 31 hours of interviews from 26 engineers to learn with whom these individuals collaborated and which actions the individuals considered helpful or unhelpful for coordination. The interviews were audio recorded, transcribed, and coded. We corroborated our interview data with a followup survey of 775 Microsoft engineers.

The interviews revealed that the most helpful behaviors between all of the teams are related to capacity, co-operation and availability. The unhelpful behaviors are related to location (primarily felt by the two distributed teams), ownership and awareness, capacity and availability. We find that problems of coordination in the distributed teams can be considered a superset of the problems of colocated teams; distributed development adds additional difficulties caused by location (i.e. time zone), email, culture, and meetings. The survey data supports the results of the interview survey; many of the problems perceived by the interviewed team members are also seen in other groups at Microsoft. We use our findings to provide a system-level perspective of the team's network of task-based interactions, organized by role and location, which highlights the effectiveness of the team's relationships. Based on our findings, we suggest concrete actions that can be taken to improve coordination between software teams.

2 The Microsoft Team

Our field study was conducted with a Microsoft software team that develops an online services product. At the time of the study, the product team was 3 years old and had grown to 300 employees from its initial 25. The team is organized into two main groups, each reporting to a different general manager, who both report to a single corporate vice president. To preserve anonymity, the first group will be referred to as Application, the second as Platform. The Application group develops the end-user application on top of the Platform group's APIs and foundation frameworks.

Originally the two groups had been a single group with a single code repository, but had been split last year due to its large size. Additionally, due to space constraints, the teams were separated into adjacent buildings. When the groups were split, their code repository was split as well; development was thereafter conducted independently. A significant number of features are developed for the Application group by distributed teams located in Boston, MA and Hyderabad, India. These two distributed teams are considered separately in the study and will be referred to as "Boston" and "India."

Technical employees at Microsoft fall into three distinct roles: Developers (Devs), Developers in Test (Tests), and Program Managers (PMs). A Dev Lead, Test Lead or PM Lead manages other members of the same role. Features are owned and developed by cross-functional teams called Feature Crews, consisting of a PM, Dev and Test, who act as peers. A PM prioritizes the features that will go into the software, and writes the high-level feature specification that Devs and Tests will use for their work. A Dev is responsible for software design, writing new features and fixing bugs. Tests translate feature specifications into test cases and manually test the software. The Application group's feature teams are composed into 5 main "product units," three located in Redmond, one in Boston, and one in India.

3 Study Methodology

The study was conducted June through August 2007. The study began with 31 one-hour interviews with 26 members of the Microsoft team, including members from the Boston and India groups. The findings from the interviews were then supported through a survey sent to a random sample of approximately 2,500 of Microsoft's technical employees (which includes developers, testers, program managers, architects, and other software engineers). The survey was filled out by 775 people, giving a very high 31% response rate overall (32% Devs, 34% Tests, 22% PMs, 45% Architects). The survey was titled "Coordination in Software Development," was anonymous, and respondents were entered into a draw for a $250 Amazon.com gift certificate.

3.1 Interviews

The interviews consisted of one-hour, semi-structured, open-ended questions based on the 'echo' method designed by Bavelas [2]. The method has recently been used in studies of organizations to examine task interactions in new product development [11], task structures of engineers [34], and manufacturing flexibility [37]. The method elicits the subject's social network of task-based interactions with the subject in the middle and nodes on the outside representing the individuals, groups or technologies with whom the subject interacts while doing his or her job. 'Echo questions' are then used to examine interactions between the subject and each of the identified nodes. The subject is asked to provide concrete examples of behaviors performed by other nodes that are helpful from the subject's point of view, and examples of behaviors that are not helpful. The exact format of the question was, "When you interact with this person, can you give some examples of things they do that help you complete your job? How does this impact your job?" By asking for specific examples of positive and negative behaviors, subjects are encouraged to provide descriptive information about actual events experienced on the job rather than ungrounded opinions or stereotypes about the behavior of others. The questions are designed to attenuate the confounding effects of perceptual biases.

In total, 13 subjects were from the Application group (6 PMs, 5 Devs, 2 Tests), 7 from the Platform group (5 PMs, 2 Devs), 3 from Boston (2 PMs, 1 Dev), and 3 from India (2 PMs, 1 Dev). Interviews were face-to-face, with the exception of Boston and India, which were done over the phone. Interviews were audio-recorded and transcribed verbatim, for a total of 460 pages of single-spaced text. The transcripts were used as the basis for this analysis.

Transcripts were coded using the NVivo7 [21] qualitative analysis software in a three-pass coding process. The unit of analysis was a helpful or unhelpful behavior that had actually occurred and affected the subject's task from their point of view. The coding categories emerged during the coding process in a manner similar to that described by Strauss and Corbin [42], that is, previous similar codes were revisited and revised in the light of recent codes, and specific behaviors were grouped under broader categories of behaviors which emerged from the data. The coded content were instances of *unique* behaviors from an individual's point of view, therefore the counts of codes in a category gives a rough indication of the importance (or weight) of that category [11]. The analysis method codes the unique behavior reported, rather than the number of words or the number of instances to which the behavior is referred.

The following is illustrative of the interview and subsequent coding process.

> Subject: Well, both [Platform PMs] are fairly responsive, so they keep up-to-date on their email. That's fairly rare for program managers, so I like that.
>
> Interviewer: Can you give me an example of how that would impact your job?
>
> Subject: Well, everything's time driven, so you know how when you have so many things in your mind, things just start falling out. If it doesn't come back for a week, I don't remember it for a week, so then things slip behind and it's only when they respond back to you, or someone yells at you for not doing something, that's when you're reminded of it. So when they're responsive, things just not only move faster but I can keep track of it better.

This quote was coded under the categories of: **helpful**, **availability**, **email**; as well as the role reporting the behavior and the role that performed the behavior. Table 1 presents the 20 most frequent codes (out of a total of 41) with a short explanation of each.

3.2 Survey

Following the interviews, we created a survey to corroborate some of our findings aobut dependencies and interpersonal feelings between teams. The survey questions were divided into three sections: demographics, facts about how coordination occurs, and perceptions of coordination quality. For space considerations we will focus on the last section.

We asked eleven 6-point frequency questions about the recipients' perceptions of how well coordination was practiced within their team and its dependencies. A scale of "All of the time, Frequently, Occasionally, Rarely, Almost Never, N/A" was used. Five 5-point Likert scale questions asked how the respondent felt about the relationships they had with their dependencies. In this question, we used the standard "Strongly Agree, Agree, Neutral, Disagree, Strongly Disagree" scale. A final three 6-point frequency questions addressed distributed development. These questions asked respondents if they interacted with teams in other time zones and about the quality of communication and coordination between such teams. The results of these questions are presented in Table 2.

Code	#Help	#Unhelp	Description
Capacity	41	30	Ability (available time, capable skill, etc.) to complete the task
Location	6	60	Behavior amplified by being in a different physical location
Status & Change	11	53	Communicating status or change of a work item to a dependency
Availability	24	29	Responsiveness to email, IM, phone, or being around when needed
Cooperation	34	18	Willingness to help
Ownership & Awareness	5	31	Maintaining mental map of who is responsible for what
Email	10	20	Email itself, or as an amplification of helpful or unhelpful behaviors
Social Networking	13	11	Knowledge of who knows what or how to find this out
Division of Labor	4	19	How work has been split across functions & teams (task structure)
Meetings	10	11	Stand-up, status, bug triage, allowing others to participate or observe
Priorities	2	19	Communication of a team's priorities and their decision process
Bureaucracy	1	16	Mngmt decisions that impact tasks and task structure
Clear Communication	6	10	Well thought-out, logical, careful communication over any modality
Culture	1	15	Accepted norms of behavior which differ between groups and locations
Comm. with Dependencies	0	16	Timely and accurate communication with your direct dependents
Visibility	0	16	Keeping others aware of schedule and progress on work items
Tools	3	12	SE tools as a source, or amplification, of helpful/unhelpful behaviors
Dependency Management	4	8	Tools and techniques for awareness of, or dealing with dependencies
Growth	2	9	Behaviors due to, or amplified by, rapid increase in size of teams
Follow Up	3	7	After a request is made, requester proactively requests status updates

Table 1: Top 20 codes arranged by frequency of unique behaviors reported. The # Help and # Unhelp are the frequency counts for Helpful and Unhelpful behaviors per category.

4 Results

The results of the interviews are presented within our thematic framework of communication, capacity and cooperation. We present the categories that impacted the subject's tasks the most, that is, the codes with the highest frequency. We briefly describe the category and give a sample quote that was coded into that category. The reader should refer to Table 1 during the discussion for counts of helpful and unhelpful behaviors in each category. Counts are of total behaviors as reported by all roles in all locations. Although most of the following behaviors are also reported by members of Boston and India, some categories emerged from the analysis which are specific to distributed teams. These issues are presented in Section 4.4.

4.1 Communication

Communication was not used as a code directly, rather it was a superordinate category encompassing more specific codes, such as **status & change** and **ownership & awareness**, explained in detail below.

4.1.1 Status & Change

The first significant category of helpful and unhelpful behaviors is the communication of Status & Change, which is defined as: updating a dependency on the status of a task or decision they depend on, and communicating a change which will vitally impact the dependency's tasks. An Application Dev Lead said, in reference to Framework Devs:

> I would appreciate a week-before update to know if they'll make the date, if they'll be early or late, or where they are. Particularly for example if you're changing the API...We've had instances where they've updated the API and they though it shouldn't impact anyone, but they change it and something breaks.

This category is a significant portion of all comments (9%), and the overwhelming majority of comments (83%) were unhelpful, indicating that there are many more instances where Status & Change are incorrectly communicated to dependencies, and this miscommunication negatively impacts the subject's tasks. 79% of our survey respondents

Coordination in Software Development Survey Questions	% Answering
I need to ping the people I depend on for status.	49% Frequently or more
My dependencies proactively communicate their status to me.	34% Frequently or more
Work items I depend on are changed without my knowledge.	67% Occasionally or more
My team's responsibilities overlap with the teams I depend upon.	68% Occasionally or more
My team has problems deciding which team has responsibility for a work item.	47% Agree or Strongly Agree
I want to know more about the status of the team I depend on.	79% Agree or Strongly Agree
I find it difficult to get a team I depend on to implement a change I require.	25% Frequently or more
I need to maintain constant contact with my dependent team in order to get what I want.	78% Agree or Strongly Agree
I feel that having personal connections with teams I depend on is helpful to me.	88% Agree or Strongly Agree
I would like to improve the relationship my team has with our dependent teams.	84% Agree or Strongly Agree
It is difficult to work with dependent teams because our commitments are not aligned.	60% Agree or Strongly Agree
I depend on teams in other time zones, or they depend on me.	44% Frequently or more
Communication is difficult with teams in other time zones.	64% Occasionally or more
The benefits of working with distributed teams outweighs the difficulties.	52% Frequently or more

Table 2: Survey results, 775 respondents, 31% response rate.

want to know more about the status of the teams they depend on, and 67% report that work items they depend on are changed without their knowledge occasionally or more.

Status & Change are not only with regard to technical issues, such as code and APIs, but also with regard to non-technical issues, such as scope of features, scope triage, bug triage, the prioritization (ranking) of a future version's features, or general status about the progress on a work item.

For example, a Platform PM said, when asked about unhelpful behaviors from Application PMs:

> The one thing, and it's the nature of the platform-partner interaction, I wish that they would do more of the tracking on the status rather than I going to them. I wish that there was more pulling of information from their side instead of my pushing information and status updates to them.

An Application PM said, when asked about unhelpful behaviors they do that the Platform PM considered unhelpful:

> Sometimes I do think that my job kind of becomes nagging. I want to be kept up to date, because it is my feature, I own it. Because I've spent time in this, I want to know exactly how it's coming along.

The quotes above indicate a mismatch in role expectations, which often leads to uncertainty and conflict [24]. Is it the requestor's job to ping the requestee, or is it the requestee's job to keep their dependents informed of their status? From our survey, we learned that there is no automated tool used universally across Microsoft to communicate Status & Change among dependencies; communication occurs manually and on an ad-hoc basis.

4.1.2 Ownership & Awareness

Ownership is defined as: communication about who owns what (division of responsibilities), and keeping aware of the changes in responsibilities within and between teams. The majority of reported behaviors are unhelpful, indicating that this is a significant source of coordination difficulty. Many highly impactful behaviors are not directly technical issues, but rather human to human communication issues. An Application PM said, in reference to the Platform PMs:

> Because of all the...overlap of our teams, it's hard to, one, delineate where the ownership line lies, and, two, it's redundant to have two people on two separate teams own the same thing, so that's the case where one owner needs to be decided, and how that's done is very arbitrary.

Subjects reported having trouble finding out which PM in charge of a feature, which Dev was responsible for a module, or even which group owned a portion of the product. It was reported that much time was spent trying to track down ownership and maintain awareness. 47% of our survey respondents said that their team has had problems determining which team has responsibility for a shared work item. This coding category is closely related to **social network** in Section 4.2.3.

4.2 Capacity

The category of Capacity emerged very early in the coding process to encompass a number of specific behaviors that related to "the ability to complete the task." Capacity is the largest portion of all behaviors. Unsurprisingly, a large number of helpful behaviors simply refer to an individual's ability to complete the tasks assigned to them. We discuss three specific aspects of capacity below.

4.2.1 Capacity & Availability

Availability is defined as "responsiveness to communication (IM/Email/Phone), being in the office when needed." A Platform PM said, in reference to an Application PM:

> Not that it's bad, but the one thing that could have gone better, and he acknowledges this—I've talked to him before—he doesn't answer email as much as I like. If I have questions I'll have to ping him a couple of times to get an answer.

Coded as: **unhelpful**, **email**, **availability**, and **capacity**. While the Platform PM reports that this is an unhelpful behavior that affects his ability to complete his tasks in a negative way, he recognizes that the behavior is not purposefully unhelpful. In this case, and in many others recorded, the underlying cause of the unhelpful behavior is a lack of capacity. Also note that the **email** code is actually under the communication category. This is an example of the overlapping nature of the necessary components of coordination; capacity affects communication, and vise versa.

Every interview subject commented that they experience email overload, with one PM receiving over 500 emails a day, and six others receiving more than 200 a day. Yet most subjects admitted that they use email as the primary method of 'pinging' their dependents for status, and our subjects said that they probably rely on email too heavily. 49% of our survey respondents reported having to frequently ping the people they depended on for status updates; 69% of all respondents do it by email.

4.2.2 Dependency Management & Priorities

Dependency Management is defined as "the tools and techniques for awareness of, or dealing with dependencies," and priorities is defined as, "communication of a team's priorities and their decision process to those that are affected by them." A Platform PM said, in reference to Application PMs:

> It's really easy to forget. To have things fall off your list of things that you're worried about. And only at the last minute when you're trying to deliver do people remember – oh, yeah. 'Whatever happened to this thing you said you were gonna get us?' And a lot of times you'll see people get that deer in the headlights look of 'oh, right. I need to get that for you.' It's surprising how often that happens.

Coded as: **unhelpful**, **capacity**, **dependency management**, **priorities**.
Similarly, an Application PM said:

> It's a question of priorities. It always comes down to priorities. You can move the earth if you prioritize it high enough, that's why in WWII they made the atom bomb in two years. The single most important thing that I do is prioritize.

Many behaviors, both helpful and unhelpful, ultimately related to an individual's ability to keep track of their commitments, to prioritize work items competing for their limited amount of attention. In some cases unhelpful behaviors, such as low availability or poor dependency management, are matters of overloaded capacity. Many of the subjects indicated that they had trouble tracking who was depending on them and for which tasks. Surprisingly, tool support for dependency management is currently limited.

An unexpected finding was that some subjects rely on a dependency pinging them as their tracking mechanism. Other subjects (notably the PMs in all groups) use tracking systems, originally designed for code-level bugs, to track feature requests and other non-bug work items. From our survey, we learned that the most common dependency tracking tools at Microsoft are email, the bug database, status meetings and (surprisingly) keeping a mental list.

4.2.3 Capacity & Social Network

Social Network is defined as "who you know; knowing who knows the information you need, or knowing who can get you in touch with someone who knows." An Application PM said, in reference to a Platform PM:

> [In reference to a feature that the Application group depended on that was triaged out of the Platform group's feature list.] I contacted [the Platform PM], explained the situation, and he immediately was able to reverse everybody's thinking and now this is going to be something that they are working on and we will get it done.

Coded as: **helpful**, **capacity**, **cooperation**, **social network**, and **negotiation**. Here the Platform PM needed to have the time available, the ability to do the work, and be willing to go out of their way to do something as helpful as negotiate on behalf of a member of the Application group.

It also demonstrates the importance of human factors such as social networks and negotiation in a field traditionally viewed as predominately technical. The behaviors are reported not only from roles that would be expected to deal with human issues (PM), but from technical roles as well. A Platform Dev said:

> I think all of this stuff boils down to one person relating to another person as a personal relationship as opposed to a team. I think of a couple of individuals who I talk to on a regular basis. And when one person leaves that is one of the guys who I talk to, to get help, it's hard.

An Application Test Lead said:

> I contacted their [Platform] Dev Lead and he was pretty responsive, on it immediately, debugged the problem with me and then we figured out the root cause. We had a fix within a 3 or 4 hour window and we're taking it to production now.
>
> Q: Have you interacted with this Dev Lead before?
>
> Yeah, I have and the key here is that you actually have to have a very good personal relationship built up with these folks.

88% of survey respondents felt that having a personal relationship with teams they depend on is helpful to them. 84% wish they had a better relationship with those teams.

4.3 Cooperation

Similar to Capacity, the category of Cooperation emerged early in the coding process to encompass a number of specific behaviors that related to "a willingness to help." Cooperation is the sixth most important coding category overall, however when considering only helpful behaviors it is the first most important. Subjects reporting helpful behaviors often gave examples similar to what this Platform PM said, referring to Application PMs:

> So I think the willingness on their side to understand some of the issues that we have helps us avoid a lot of escalation and stuff like that.

Simply the willingness to understand the other's point of view is a major source of helpful behaviors. There were examples of unhelpful behaviors as well. An Application PM said, in reference to a Platform PM:

> I just get a no response or a no, I can't do this, or they bounce me around to a bunch of other people. That's definitely happened quite a bit where you won't get a response for days and I have to repeatedly ping somebody over and over again.

Coded as: **unhelpful**, **cooperation**, and **availability**. As noted in the Capacity section, the unhelpful behavior may have been an indirect result of capacity overload. However, as this Application Dev notes:

> I know they have other priorities in their job. If they tell me they can't give time now, I kind of respect them. And if they say, yes I have done code profiling before, I know how this code profiler works, but I will give you time tomorrow. So I'm like, why don't you take your 15 minutes right now and save me an entire day?

	AppDev	AppPM	AppTest	PlatDev	PlatPM	PlatTest	AppTotal	PlatTotal	Redmond	Distr
AppDev	2 / 2	8 / 5	2 / 3	1 / 19	4 / 12	0 / 0	14 / 14	5 / 27	0 / 0	0 / 5
AppPM	1 / 1	0 / 0	0 / 1	1 / 1	15 / 40	1 / 0	4 / 22	16 / 41	0 / 0	3 / 19
AppTest	2 / 1	2 / 0	0 / 0	1 / 3	0 / 2	0 / 1	12 / 19	1 / 5	0 / 0	4 / 9
PlatDev	2 / 5	1 / 3	0 / 0	0 / 0	1 / 0	0 / 0	3 / 7	1 / 0	0 / 0	0 / 0
PlatPM	7 / 6	21 / 31	2 / 1	5 / 3	7 / 4	0 / 2	25 / 34	11 / 14	0 / 0	0 / 2
Boston	1 / 0	1 / 0	9 / 6	0 / 1	2 / 2	0 / 0	17 / 27	2 / 2	6 / 22	0 / 0
India	0 / 0	3 / 0	0 / 0	0 / 0	3 / 1	0 / 0	19 / 30	3 / 1	16 / 25	5 / 7
AppTotal	5 / 4	10 / 5	2 / 4	3 / 23	19 / 54	1 / 1	30 / 55	22 / 73	0 / 0	7 / 33
PlatTotal	9 / 11	22 / 34	2 / 1	5 / 3	8 / 4	0 / 2	28 / 41	12 / 14	0 / 0	0 / 2

Table 3: Frequency of Helpful / Unhelpful behaviors by role. Row is the role reporting the behavior (point of view), column is the role being reported about.

4.4 Distributed Development

Globally distributed development is an important issue for modern software companies. 44% of survey respondents say that they frequently depend on teams in other times. The results in this section demonstrate that while many of the issues in distributed development are not necessarily unique, being non-colocated may amplify their effects.

4.4.1 Location

A Boston PM in reference to teammates in Redmond:

> They end up sending us an email at 6 or 7pm Redmond time about this urgent issue and I end up having to deal with it, and it's 10pm my time, I'm at home.

Coded as: **unhelpful**, **location**, and **email**. It appears that distributed teams report similar helpful and unhelpful behaviors as their non-distributed teammates, but as in the case above, what would have been only an issue of **email** is coupled with **location**. Interestingly, none of the **location** codes are single codes, there is always a second category for the behavior. This implies that location *amplifies* the issues already experienced by colocated teams.

An unexpected finding of this category is that location-related behaviors are almost entirely unhelpful. A Boston PM:

> We miss out on a lot of those water cooler conversations...we just can't stick our head in people's offices, so what we end up doing is sending an email and the turnaround is so long for that.

While not a behavior (and thus not coded as such in the analysis), this illustrates that many of the social and informal communication mechanisms taken for granted by colocated teams are absent in distributed teams, and the absence affects task behavior. 78% of survey respondents said they needed to maintain constant contact with the teams they depend on in order to get what they wanted. This suggests that coordination, which respondents report requires constant contact with dependencies, will become more difficult with distributed teams, assuming constant contact is more difficult in distributed teams.

4.4.2 Culture

While every team has norms, distributed development may be unique in introducing significant cultural effects into the daily work of software development. A PM Lead in India, in reference to his own team:

> I have to iron out miscommunications between Redmond...and teach my team how to better communicate with them over email.

This behavior was categorized under **unhelpful**, **communication**, **location**, and **culture**. The managers in India mentioned that their employees had difficulties communicating through email to their teammates in Redmond. The end result was hurt feelings on one side and confusion on the other. For example, a Redmond employee would

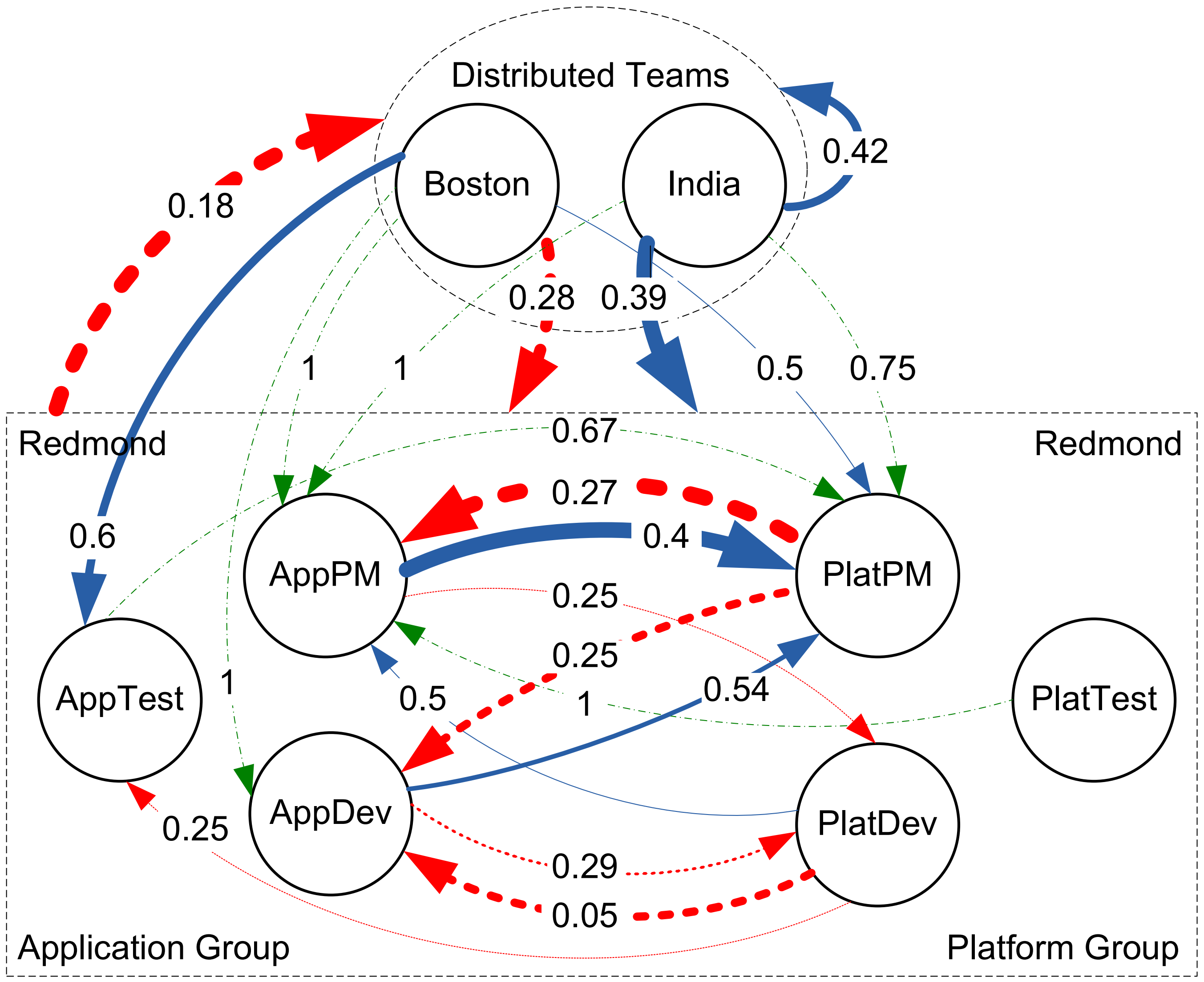

Figure 1: Network graph of the effectiveness ratio between roles.

start a communication by immediately diving into the problem, which gave the impression to the Indian colleague that they were busy and swamped with work, that this was an interruption they could do without, and that they wanted to get it over with as soon as possible. The Indian colleague would thank the Redmond employee and leave, feeling like they had just been brushed off.

5 A System Perspective

If we organize behaviors by role and by location, the interview data can be reorganized as Table 3 (as presented in Safayeni *et al.* [33]. The left column indicates from whose point of view the behavior is being reported, the top row is the role that is being reported about. The first number in a cell is the number of helpful behaviors reported, the second is the number of unhelpful behaviors reported. For example, the intersection of PlatPM row and AppPM column gives 21 / 31; the PlatPM (Platform PM) role reported 21 instances of helpful behaviors and 31 instances of unhelpful behaviors from the AppPM (Application PM) role. Redmond is an placeholder role for behaviors reported by distributed teams (India and Boston) about workers in corporate headquarters. Boston and India are combined into "Distr", as neither role reported on each other, and other roles reported on distributed teams in general. PlatTest as a point of view (left column) is absent, as we were unable to interview their members.

It is then possible to produce a systems view of the situation by organizing unhelpful and helpful behaviors by role into a network graph (Figure 1). This view allows us to see the overall health of the network of task-based interactions. The nodes of the graph represent roles, and the weight of the edge connecting the nodes (the links) is the ratio of the number of helpful behaviors divided by the total number of behaviors, called the *effectiveness ratio (ER)*. This ratio is designed to give a rough indication of the effectiveness of the interactions between roles, highlight healthy and unhealthy links, and focus discussion on links that could be learned from or improved. A green, dashed line represents a helpful relationship (ER >= 0.67), a red, dotted line represents an unhelpful relationship (ER <= 0.33), and a blue, solid line represents a neutral relationship (0.67 > ER > 0.33). The direction of the link indicates the point of view; if the arrow is pointing to a node, the link is from that node's point of view. The link direction can be thought of as variety [1] being sent from a sender node to a receiver node, from the receiver's point of view. The weight (thickness) of an edge is a rough approximation of the importance of the link, as measured by the frequency of total behaviors attributed to node Y by node X.

One finding from the system perspective is the significant asymmetry in the role relationships. Specifically the AppPM—PlatPM link and the AppDev—PlatPM link. In both of these relationships, the Application team members view the relationship as less helpful than the Platform team members. This can be partially explained by the Platform—Application task structure: the Application team consumes the output of the Platform team, but the Platform does not directly consume the output of the Application team. The dependency is sequential rather than reciprocal, and the one-way dependency causes the Application team members to feel like they are continually asking the Platform to do work, without being able to offer anything in return. An Application PM said, in reference to a Platform PM:

> So, she's very helpful, but I feel like she definitely helps me more than I help her. But there really aren't things that we provide to [the Platform group]. I mean that's not really how—it's sort of weird because the relationship isn't like a two way—we're consuming their stuff, we're not really giving them anything. So there's nothing that they really need from us or want from us. We want stuff from them and we expect stuff from them.

Also surprisingly, the only intergroup helpful (green, dashed) links are from people who reported very few behaviors from the sender (three or less). Significant inter-group links (over five behaviors) are neutral (blue, solid) or unhelpful (red, dotted), indicating room for improvement in the interaction. Although not the focus of the study, some intra-group behaviors were reported, which tended to be positive.

As discussed in the Capacity section, unhelpful behaviors are rarely intentional acts of malice, but are often the result of a lack of capacity. Therefore, the unhelpful (red, dotted) links should not be interpreted as intentional, or perceived as negative relationships between roles. The benefit of this perspective is to recognize asymmetries in the relationships; interactions in which behaviors are perceived to be unhelpful by one person but are either not recognized as unhelpful by the sending person, or not even recognized as significant by the sending person.

6 Threats to Validity

We had an unsymmetrical sample for interviews due to difficulties in obtaining interviews with the Platform group. Due to the unsymmetrical sampling, the data in Section 5 should be considered lopsided, with more behaviors reported by the Application group than the Platform group. Asymmetrical data affects the link weight, that is, the total number of behaviors reported for the interaction link. The link valence is less affected, as the valence is a ratio. The data in Section 4 should be unaffected by the asymmetry, as behaviors are grouped by role across group and location.

The interview data is gathered from the subject's point of view, and hence cannot be considered an objective measure of effect on task performance. It is possible that the subject misperceives behaviors from their interactions with others. However, if misperceived behaviors affect the subject's tasks, then they are significant, at least from their point of view. The study would certainly benefit from parallel measures of impact on task performance, perhaps using software engineering metrics, if available at the interpersonal interaction level of analysis.

Our study was conducted at Microsoft Corporation; while we imagine its results apply broadly to software developers at other companies, studies at other sites would be useful to highlight behaviors which may be affected by Microsoft norms and culture.

Although the employees were experienced in other product groups, this particular group was 3 years old, relatively young compared to some others. Further, some of our findings could be due to the group's growing pains.

7 Impact on Process and Tool Design

Generally speaking, there are two methods for increasing the ability of a person to do his job: increase his variety handling ability (capacity), or decrease the variety from the environment (amount of work or unhelpful behaviors from other people) [1, 37]. It may be possible to increase a person's capacity through to use of software engineering tools, and it may be possible to reduce the environmental variety through process redesign, organizational-level change, or interaction-level change.

7.1 Communication

The size of the Status & Change and Ownership & Awareness category indicates that significant improvement can be made with process or tool support. Many of the unhelpful behaviors in this study involved coordination of non-technical issues, such as those that are traditionally assumed to occur between program managers, upper management, designers, user experience engineers, planners, and marketing. At the time of the study, the Platform team had created a role whose job was to put all interteam dependencies on an internal web-based tool. Regrettably, the tool required manual updating and notification of dependencies and deadlines. A tool to automate dependency tracking (by some means), provide automatic notification of dependency due dates, and possibly automatic notification of dependency change based on user-specified dependency "interests" would consume fewer personnel resources and be more useful for team members.

7.2 Capacity

Capacity was the largest category, but may be the most difficult to improve. As noted in previous sections, categories of unhelpful behaviors such as **availability**, **dependency managment** and **priorities** are often, at their root, a result of insufficient capacity. While it may not be possible to directly increase an engineers's capacity, it would be much easier to decrease the external variety impacting an engineer. A major source of external variety identified in the interviews was email. Possible solutions include better email filtering tools, reducing the number of automated system messages sent through email, and instituting policies aimed at reducing frivolous email traffic and needless carbon copying. For example, an informal policy of "no meetings on Friday" was recently implemented by the Application group in an effort to reduce environmental variety generated by meetings, although this particular solution may have the unintended consequence of increasing unhelpful behaviors associated with **status & change**, among others.

The unhelpful behaviors coded as **ownership & awareness** may be an unavoidable effect of large-scale software development. Tools should be designed specifically to aid awareness in large-scale development: which PM is in charge of which feature, which Dev and Test are the owners of which code, and when do these ownerships change and why? Perhaps most difficult, however, is engineering the transfer of this knowledge across team boundaries and code bases. The problems seem easy to fix, but they are deceptively complex when dealing with large-scale development and globally distributed teams.

7.3 Cooperation

As noted in the cooperation section, many helpful behaviors were simply the result of one person willing to go out of their way to help another; or in other words, variety is handled by one person, freeing up another person's capacity. Unfortunately workers are overloaded in the first place and often cannot cooperate, for fear that their work won't get completed. But often a small amount of work by one person will unblock another, saving hours or days of frustrating deadlock. One possible process improvement then, is to implement a policy: helping teammates is the highest priority, at least for a short amount of time. Of course this policy might have unintended consequences, but the net benefit may be positive.

7.4 Distributed Development

Data from the interviews indicated that cultural understanding is vitally important to effective distributed teams. A frequent suggestion was to increase the number of Redmond employee visits to India. In the words of a PM in India:

> We've had some key leads from Redmond not just visit here, but work here out of India for two weeks. Once anybody goes through that experience, they will never forget India.

Other suggestions included placing pictures of their coworker's faces in Outlook, or video-recording Redmond meetings for viewing during work hours in distributed locations. A reasonable suggestion, from the point of view of a PM in India, would be to vary the schedule of cross-team meetings, so that more are held during work-time in India, if only in proportion to the number of people in each location.

8 Related Work

In this section we summarize the work on coordination in software development that influenced the development of our study.

8.1 Coordination in Software Teams

The closest work to our own is a study by de Souza and Redmiles of how individual software developers manage dependencies between teams [9]. They used an ethnographic approach to study two companies to discover work practices and rationales for dealing with changes to software. They introduce the concept of impact management, which shows how developers minimize the impact of their work on others and at the same time minimize the impact of others' work on their own. Their results point at the changing nature of the network of developers who impact one another, the size of this network, and strategies developers use in practice to minimize their effects on others.

Another related study, by Herbsleb and Grinter [17], explores geographically distributed software development in a project based on teams working in Germany and UK at Lucent Technologies. Based on a total of 18 interviews, the prominent coordination factors were integration of the system built by the teams, specification of programming interfaces, process mechanisms, and documentation. Consequently, the primary barriers to team coordination were lack of unplanned contact, knowing the right person to contact about specific issues, cost of initiating the contact, effective communication, and lack of trust. Herbsleb and Grinter provide recommendations based on their empirical case study for organizations with respect to communication barriers and coordination mechanisms.

From a theoretical perspective, Herbsleb and Mockus [18] formulate and evaluate an empirical theory (of coordination) toward understanding engineering decisions from the viewpoint of coordination within software projects.

Cataldo et al. [5] compares the actual coordination of members of a software team with a measure of suggested coordination: the cross-product of people assigned to a modification request with task dependencies. Two tasks were marked as dependent if source code files associated with the task were modified at the same time in the source control system. Multiplying the task assignment matrix with the task dependency matrix produces a people by task matrix which suggests the set of tasks each person should be aware of when working on their own task. Multiplying again by the transform of the task assignment matrix produces a person to person mapping indicating which developers should be communicating and coordinating in order to get their tasks done properly. They compared actual coordination with their predictions of coordination requirements in a 114 person, 8 team, 3 locations project in a data storage company and show that task performance increases when these two measures align and coordination requirements are contained within a team. Geographical closeness also correlated with positive task performance.

Purvis et al. [32] describe an experiment in distributing a software project between students in Germany and New Zealand. One salient problem they ran into was determining who on the teams had responsibility for issues concerning common components. They suggest clarifying these overlapping concerns early on in development. Sandusky and Gasser describe a study of bug fixing in open source projects as an exercise in negotiation between coordinating software developers [35].

8.2 Communication in Software Teams

Fred Brooks, in the classic Mythical Man-Month [13], states that in scheduling disasters, functional misfits and system bugs arise from a lack of communication between different teams. Gutwin et al. [15] observed the requirements and mechanisms for group awareness in three open source system (NetBSD, Apache httpd, and Subversion). They observed that open source developers maintain a general awareness of the team and knowledge about people they plan to work with. The primary means of awareness were text-based communication mechanisms: mailing lists and chat tools.

Curtis et al. [6] asserts that since the software is created by people, the study of its creation must be analyzed as a behavioral process. He suggests studying software development at several organizational levels, including group dynamical and organizational, as we have done here. In Curtis et al.'s interview-based study of 19 software projects, the most obvious problems were due to poor knowledge flow across people, changing requirements, and communication and coordination breakdowns. Groups reporting to different chains of command often had trouble disseminating knowledge across the boundaries. Communication problems occured during product lifecycle phase transitions when control was transferred from one software team to the next. Within the team, status updates were difficult to manage as team sizes grew. Many of the problems that Curtis et al. attributes to communication also have analogues in capacity and cooperation, which we will discuss below.

Kraut and Streeter [27] studied communication and coordination via survey in the software department of a research and development company. Integrating data from 65 software mid-sized (15 people median per team) projects, they found developers had a distinct preference for coordinating via communication with peers, project schedules, bug tracking, customer testing, and requirement and design reviews. Frequent, informal communication was common even in larger projects that had more formal meetings, even in those which had formal meetings that respondents found valuable. When asking for help, other team members were the main and best source of answers. Kraut and Streeter use their survey results to develop a complex model to predict successful coordination. They find that their results about coordination in large software development settings concur with results about coordination issues faced by other types of organizations.

Ko et al. [26] report on an observational study of 17 software developers at Microsoft conducting maintenance tasks. One of the most common information needs of developers they found was status: "what have my co-workers been doing?" For many of the other information needs, answers were most often found by talking to co-workers, which emphasizes the importance of availability to support software team coordination.

Perry et al. showed that unplanned interactions between software developers occupied an average 75 minutes of each developer's day [31]. In addition, each developer met with, in person, an average of 7 people per day. Wu et al. [47] studied a software team that communicated 15 times per day for an average of 124 minutes per day, and usually in groups of 3. LaToza et al. [28] surveyed software developers across Microsoft and found that developers preferred face-to-face meetings to communicate with their colleagues. They found that an average of 8.4 of these meetings occurred per week within a team, but only 2.6 a week with people outside the team. They conclude that this is a feature of the boundaries set up by teams to insulate themselves from others, however we find this communication preference hinders communication between teams that are linked by dependencies, but not co-located. de Souza et al. [7] finds that boundary objects such as APIs can have negative consequences on coordination between teams by encouraging isolation and hindering awareness between API producer developers and consumer developers.

Seaman finds evidence to support the hypothesis that code review meeting participants require more effort to communicate when the group includes a few organizationally distant members, than when all of the group members are organizationally distant from one another [39]. Seaman hypotheses in a later study of code inspections [40] that the number of defects reported and the number of global issues discussed should vary inversely with team member familiarity, and physical and organizational closeness. We find this to be the case for many kinds of issues when looking at the interactions between software teams in Redmond, Boston and India.

8.3 Improving Coordination

Many researchers have proposed treatments, both software and process, to improve coordination in software development. de Souza et al. [8] identify a critical communication barrier caused by source code control systems that strive to isolate developers' code (and thus developer interactions). Dependencies between section of code cause social dependencies between the developers who own the code. These developers may be unaware that they are working on the dependent code at the same when using optimistic concurrency models of source control or a system of code branches that are associated with private and public purposes; de Souza reports that in his study that coordination consisted of informal emails warning of pending checkins and code reviews within teams – the first mechanism is easily forgotten in haste and the second does nothing to warn outside groups of pending and potentially conflicting changes.

Several tools to promote developer awareness with respect to source code have been prototyped, such as Ariadne [45], FastDash [3], Elvin [12], Jazz [20], Palantir [36], TUKAN [38] and Night Watch [30]. Storey et al. [41] provides a useful framework for designing and comparing awareness tools like these, and Grinter [14] presents a study of one source code control coordination tool's use in a software company. Herbsleb and Mockus designed ExB, an expertise visualization browsing tool that lets developers find experts associated with some part of their source code [29]. However, these tools are all code-centric; none support coordination over a procedure or social process. Halverson et al. [16] introduced the Social Health Overview, an awareness tool to visualize the status of work items in a bug database and improve coordination over their resolution.

To improve knowledge flow across members of a team, and potentially across teams, Team Tracks [10] and Mylyn (neé Mylar) [25] gather IDE navigation history from experts and share them to new users either explicitly through a document model (Mylyn) or implicitly through a recommender system to guide newcomers to view situations as the previous users have already done.

Supporting unplanned interactions in non-collated teams has also been a rich avenue for research. Media spaces supported by audio and video feeds can link separated groups [4], providing peripheral awareness, but suffer from a lack of situational awareness – the link is always on, whether you want to talk or not. Online chat systems and instant messaging serve as popular forms of synchronous communication between separated team members; they provide awareness and a communication medium, but can be offputting to use when you need to communicate with people you do not know personally. Piazza [22] tries to combine the awareness of instant messaging with a notion of ”nearbyness” and chat rooms for contextualizing discussions. Some companies are experimenting with virtual worlds such as Second Life to bring non-collocated team members together for meetings and impromptu discussions, which some research [23] suggests will be effective.

Radical collocation has been proposed and studied by Teasley et al. [43, 44]. By placing entire teams of developers into a single ”warroom,” awareness, coordination and availability are enhanced and result in great productivity gains. For teams distributed across time zones, evolving an informal hierarchical structure, such as a point person who takes charge of communicating with a remote team, has been found to ease coordination [19].

9 Conclusion

Our view of coordination helps reveal how communication, capacity, and cooperation interplay to affect successful software development projects. Our study finds that many of the behaviors that impact engineers most are not directly technical issues, such as code and APIs, but rather coordination issues. These are commonly referred to as “human aspects” of software development, and they appear, from the data gathered in this study, to be highly significant to the software engineer's tasks. We show that taking a systems view of the situation reveals unanticipated asymmetries in the relationships between roles, and highlights interactions that may benefit most from improvement. We propose several process and tool changes to address each aspect of our coordination model, that, when enacted, are predicted to improve interaction effectiveness. A new study lays ahead to measure the effects of our changes.

10 Acknowledgments

We thank the members of the four software teams at Microsoft who participated in our interviews. We also thank Rob DeLine for his helpful comments on this paper.

References

[1] W. R. Ashby. *An Introduction to Cybernetics.* Chapman & Hall, London, UK, 1956.

[2] A. Bavelas. A method for investigating individual and group ideology. *Sociometry*, 5:371–377, 1942.

[3] J. T. Biehl, M. Czerwinski, G. Smith, and G. G. Robertson. Fastdash: a visual dashboard for fostering awareness in software teams. In *Proceedings of CHI*, pages 1313–1322, San Jose, CA, 2007. ACM Press.

[4] S. A. Bly, S. R. Harrison, and S. Irwin. Media spaces: bringing people together in a video, audio, and computing environment. *Communications of the ACM*, 36(1):28–46, 1993.

[5] M. Cataldo, P. A. Wagstrom, J. D. Herbsleb, and K. M. Carley. Identification of coordination requirements: implications for the design of collaboration and awareness tools. In *Proceedings of CSCW*, pages 353–362, Banff, Alberta, Canada, 2006. ACM Press.

[6] B. Curtis, H. Krasner, and N. Iscoe. A field study of the software design process for large systems. *Communications of the ACM*, 31(11):1268–1287, 1988.

[7] C. R. B. de Souza, D. Redmiles, L.-T. Cheng, D. Millen, and J. Patterson. How a good software practice thwarts collaboration: the multiple roles of apis in software development. In *Proceedings of FSE*, pages 221–230, Newport Beach, CA, 2004. ACM Press.

[8] C. R. B. de Souza, D. Redmiles, and P. Dourish. "breaking the code", moving between private and public work in collaborative software development. In *Proceedings of GROUP*, pages 105–114, Sanibel Island, FL, 2003. ACM Press.

[9] C. R. B. de Souza and D. F. Redmiles. An empirical study of software developers' management of dependencies and changes. In *ICSE '08: Proceedings of the 30th international conference on Software engineering*, pages 241–250, New York, NY, USA, 2008. ACM.

[10] R. DeLine, M. Czerwinski, and G. Robertson. Easing program comprehension by sharing navigation data. In *Proceedings of VL/HCC*, pages 241–248, Washington, DC, 2005. IEEE Computer Society.

[11] P. R. Duimering, B. Ran, N. Derbentseva, and C. Poile. The effects of ambiguity on project task structure in new product development. *Knowledge and Process Management*, 13(13):1–13, 2006.

[12] G. Fitzpatrick, P. Marshall, and A. Phillips. Cvs integration with notification and chat: lightweight software team collaboration. In *Proceedings of CSCW*, pages 49–58, Banff, Alberta, Canada, 2006. ACM Press.

[13] J. Fred P. Brooks. The mythical man-month. In *Proceedings of the international conference on Reliable software*, page 193, New York, NY, 1975. ACM Press.

[14] R. E. Grinter. Using a configuration management tool to coordinate software development. In *Proceedings of COCS*, pages 168–177, Milpitas, CA, 1995. ACM Press.

[15] C. Gutwin, R. Penner, and K. Schneider. Group awareness in distributed software development. In *Proceedings of CSCW*, pages 72–81, Chicago, IL, 2004. ACM Press.

[16] C. A. Halverson, J. B. Ellis, C. Danis, and W. A. Kellogg. Designing task visualizations to support the coordination of work in software development. In *Proceedings of CSCW*, pages 39–48, Banff, Alberta, Canada, 2006. ACM Press.

[17] J. D. Herbsleb and R. E. Grinter. Splitting the organization and integrating the code: Conway's law revisited. In *Proceedings of ICSE*, pages 85–95. IEEE Computer Society Press, 1999.

[18] J. D. Herbsleb and A. Mockus. Formulation and preliminary test of an empirical theory of coordination in software engineering. In *Proceedings of ESE*, pages 138–137, Helsinki, Finland, 2003. ACM Press.

[19] P. Hinds and C. McGrath. Structures that work: social structure, work structure and coordination ease in geographically distributed teams. In *Proceedings of CSCW*, pages 343–352, Banff, Alberta, Canada, 2006. ACM Press.

[20] S. Hupfer, L.-T. Cheng, S. Ross, and J. Patterson. Introducing collaboration into an application development environment. In *Proceedings of CSCW*, pages 21–24, Chicago, IL, 2004. ACM Press.

[21] Q. S. R. International. NVivo 7. `http://www.qsrinternational.com/ products_nvivo.aspx`.

[22] E. A. Isaacs, J. C. Tang, and T. Morris. Piazza: a desktop environment supporting impromptu and planned interactions. In *Proceedings of CSCW*, pages 315–324, Boston, MA, 1996. ACM Press.

[23] P. Jeffrey. Forum contact space: serendipity in the workplace. In *CHI extended abstracts*, pages 331–332, The Hague, The Netherlands, 2000. ACM Press.

[24] D. Katz and R. L. Kahn. *The Social Psychology of Organizations.* Wiley, New York, 2 edition, 1978.

[25] M. Kersten and G. C. Murphy. Using task context to improve programmer productivity. In *Proceedings of FSE*, pages 1–11, Portland, OR, 2006. ACM Press.

[26] A. J. Ko, R. DeLine, and G. Venolia. Information needs in collocated software development teams. In *Proceedings of ICSE*, pages 344–353, Washington, DC, 2007. IEEE Computer Society.

[27] R. E. Kraut and L. A. Streeter. Coordination in software development. *Communications of the ACM*, 38(3):69–81, 1995.

[28] T. D. LaToza, G. Venolia, and R. DeLine. Maintaining mental models: a study of developer work habits. In *Proceedings of ICSE*, pages 492–501, Shanghai, China, 2006. ACM Press.

[29] A. Mockus and J. D. Herbsleb. Expertise browser: a quantitative approach to identifying expertise. In *Proceedings of ICSE*, pages 503–512, Orlando, FL, 2002. ACM Press.

[30] C. O'Reilly, P. Morrow, and D. Bustard. Improving conflict detection in optimistic concurrency control models. In *Proceedings of SCM*, pages 191–205, Portland, Oregon, 2003.

[31] D. E. Perry, N. Staudenmayer, and L. G. Votta. People, organizations, and process improvement. *IEEE Software*, 11(4):36–45, 1994.

[32] M. Purvis, M. Purvis, and S. Cranefield. Educational experiences from a global software engineering (gse) project. In *Proceedings of ACE*, pages 269–275, Dunedin, New Zealand, 2004. Australian Computer Society, Inc.

[33] F. Safayeni, P. R. Duimering, K. Zheng, N. Derbentseva, C. Poile, and B. Ran. Requirements engineering in new product development. *Commun. ACM*, 51(3):77–82, 2008.

[34] F. Safayeni, A. Yu, L. Purdy, and E. Lee. Assessing the potential for e-mail for engineers: case study. *Management in Engineering*, 8(4):346–361, 1992.

[35] R. J. Sandusky and L. Gasser. Negotiation and the coordination of information and activity in distributed software problem management. In *Proceedings of GROUP*, pages 187–196, Sanibel Island, FL, 2005. ACM Press.

[36] A. Sarma, Z. Noroozi, and A. van der Hoek. Palantír: raising awareness among configuration management workspaces. In *Proceedings of ICSE*, pages 444–454, Portland, Oregon, 2003. IEEE Computer Society.

[37] J. Scala, L. Purdy, and F. Safayeni. Application of cybernetics to manufacturing flexibility: a systems perspective. *Manufacturing Technology Management*, 17(1):22–41, 2006.

[38] T. Schümmer and J. M. Haake. Supporting distributed software development by modes of collaboration. In *Proceedings of ECSCW*, pages 79–98, Bonn, Germany, 2001. Kluwer Academic Publishers.

[39] C. B. Seaman. Communication costs in code and design reviews: an empirical study. In *Proceedings of CASCON*, page 34, Toronto, Ontario, Canada, 1996. IBM Press.

[40] C. B. Seaman and V. R. Basili. An empirical study of communication in code inspections. In *Proceedings of ICSE*, pages 96–106, Boston, MA, 1997. ACM Press.

[41] M.-A. D. Storey, D. Čubranić, and D. M. German. On the use of visualization to support awareness of human activities in software development: a survey and a framework. In *Proceedings of SoftVis*, pages 193–202, St. Louis, MO, 2005. ACM Press.

[42] A. Strauss and J. Corbin. *Basics of Qualitative Research: Grounded theory, procedures and techniques.* Sage, Newbury Park, CA, 1990.

[43] S. Teasley, L. Covi, M. S. Krishnan, and J. S. Olson. How does radical collocation help a team succeed? In *Proceedings of CSCW*, pages 339–346, Philadelphia, PA, 2000. ACM Press.

[44] S. D. Teasley, L. A. Covi, M. S. Krishnan, and J. S. Olson. Rapid software development through team collocation. *IEEE Transactions on Software Engineering*, 28(7):671–683, 2002.

[45] E. Trainer, S. Quirk, C. de Souza, and D. Redmiles. Bridging the gap between technical and social dependencies with ariadne. In *Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange*, pages 26–30, San Diego, California, 2005. ACM Press.

[46] G. O. Wiredu. A framework for the analysis of coordination in global software development. In *Proceedings of GSD*, pages 38–44, Shanghai, China, 2006. ACM Press.

[47] J. Wu, T. C. N. Graham, and P. W. Smith. A study of collaboration in software design. In *Proceedings of ISESE*, page 304, Washington, DC, 2003. IEEE Computer Society.