Effecting Change: Coordination in Large-Scale Software Development

Andrew Begel[†]
Microsoft Research
One Microsoft Way
Redmond, WA, USA
andrew.begel@microsoft.com

ABSTRACT

Large-scale software development requires coordination within and between very large engineering teams, each of which may be located in different locations and time zones. Numerous studies, and indeed, a whole conference (ICGSE), are dedicated to discovering the causes of problems with distributed development in the software industry. Microsoft has long had product teams too large to be considered co-located, even when sitting in neighboring buildings on the same campus. Recently, it has been expanding its engineering workforce into India and China, and our research is showing that Microsoft is encountering many of the coordination problems that go along with differences of location, time zone, and culture. As we go forward, our research has been changing from learning about the problem to experimenting with solutions. What are the best practices for improving coordination? Can they be applied to all software teams? How does one move past simple readings of research results towards effective intervention?

Categories and Subject Descriptors

K.6.3 [**Software Management**]: [Software process]

General Terms

Management

Keywords

Coordination, Distributed Development

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 [8]. Many researchers have studied professional software development teams empirically to gain greater understanding of how software development processes, tools, and people impact coordination [4, 10, 12]. 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.

Microsoft is considered to be one of the largest software companies in the world. It employs tens of thousands of software engineers who work on hundreds of interdependent products. Microsoft is no stranger to coordination problems between product teams, but only in the last few years has the magnitude and types of these problems changed. Microsoft has been expanding its development overseas, starting in Europe, and continuing in India and China. Time zone and location differences are starting to play an increasing role in shaping how product teams work on their software.

2. COORDINATION STUDY

To better understand the particular coordination problems faced by distributed software teams, we conducted an interview-based study of one product team at Microsoft. The team is 3 years old and is made up of around 300 software engineers divided between Redmond, WA, Boston MA, and Hyderabad, India. It is organized into two main groups — one works on a customer-visible application, and the other works on an internal platform library used by the application team. The software product itself is composed of loosely-coupled components, one created by the team in Boston, and one by the team in Hyderabad. The other components are built in Redmond, by teams from both groups working out of two neighboring buildings.

We interviewed 26 members of the teams using questions that were designed to record the work-related social network of each interviewee, and elicit a list of helpful and unhelpful concrete events that had occurred in the last several months of each relationship. We classified each action into into four areas: communication, capacity, cooperation, and distribution.

The communications area concerned meetings (individual and group), awareness, communications modalities, frequency and quality. This is an example quote from one of our interviewees, "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 thought it shouldn't impact anyone, but they change it and something breaks."

The capacity area covered the amount of work each engineer had

[†]This research was conducted in collaboration with Christopher Poile from the University of Waterloo, Nachiappan Nagappan from Microsoft Research, and Lucas Layman from North Carolina State University.

CHASE'08, May 13, 2008, Leipzig, Germany.

and their availability. An example quote from this area is "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."

Cooperation covered dependency management, division of labor, and willingness to help. "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?"

Location and time zone issues made up the distribution area. "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."[1]

Combined, the four areas mark out a space to measure good and bad qualities of coordination. Summarizing the results, we found that many of the issues mentioned by different team members were highly asymmetric. That is, analyzing the relationship between the two groups, the issues that one group perceived as harming inter-group coordination were not the same issues that the other group felt were important. Interactions related to capacity and cooperation were cited frequently by all team members as having a positive impact on overall coordination. The remote teams, Boston and Hyderabad, were negatively impacted by location, meetings, and email problems. The team in Hyderabad reported additional problems caused by cultural differences between them and Redmond. The Boston team reported additional problems with miscommunication between them and Redmond.

The issues that negatively affected coordination between the platform library and applications teams in Redmond focused on poor communication about awareness, status, and changes. Furthermore, the platform group felt additional problems with low visibility to the application group and thought that the application group often neglected to follow up on work tasks. The application group was significantly impacted by location problems, division of labor, availability, and prioritization, all stemming from its relationship with the platform library group.

3. TREATMENTS

Once the study surveying the product team's problems was finished, we encountered a problem of our own. What was the most effective way to improve coordination between troubled software development teams? We had a list of problems and data that demonstrated correlations between various helpful and unhelpful actions and people's feelings about those actions. We found potential ideas for solutions from the literature, from surveys we conducted to find best practices (or at least attempted to find best practices) at Microsoft [1], and even invented a few on our own.

In this section, we describe the situations that caused the most significant coordination problems, list a bevy of solutions that we proposed to our studied software team in several meetings with the team's management, and paraphrase their less than enthusiastic response to our ideas. We are not saying that the team was unwilling to let us help them, more that our solutions, however compelling in their simplicity, lack pragmatic applicability to the real-life conditions in which product teams find themselves.

[1]While the example comments above are all negative, there were many positive comments.

3.1 Location

Location Problems Many informal meetings happen in hallways. Status is communicated, ownership is negotiated, and decisions are made in these hallway meetings. Teams whose members are not in the same buildings do not participate in hallway meetings, which puts them at a disadvantage.

Location Solutions Try not to make decisions in the hallways. Send status updates electronically. Make sure that those affected by the decisions feel like they have a chance to participate in the process, even if they were not there.

These solutions are difficult for product teams to accept. It is very convenient to convene a set of people in a hallway in order to bring everyone up-to-date and make decisions. This is perceived by each individual to be true whether the team is made up of five people or five hundred people. Should hallway meetings be minimized? The practice is part of baseline human nature and not so easily curtailed.

Sending status electronically means that someone has to summarize a conversation, whether it be informal (e.g. in the hallway), or formal (at a pre-announced time and place). In a formal meeting, a scribe can take notes but then has to post them. Sometimes, if he is a lead, he stores them on his own computer. Sometimes, he will email them out to the meeting participants. Sometimes, the meeting notes will get posted to a public share where they can be viewed even by people who did not attend the meeting. Intuitively, it appears that posting the notes to a share ought to be the right solution for working with distributed teams, but it turns out to not always apply. Some parts of the meeting can be sensitive, and while it is fine for the people who were at the meeting to read the notes, those parts may not be fit for more general distribution. It feels like a lot of extra work to filter parts of meeting notes (often just a Word or Excel document) to ensure visibility by a select set of people.

How can we make a team member who is not present at a decision feel like they are part of the process? This turns out to be quite difficult. The general impression of team members not co-located with the main body of engineers is that many discussions happen without them and many decisions that affect them are made without their input. Again, this is an expediency issue. While some decisions are monumental enough to require all of the team's management to sign off on them, most are small, everyday decisions which are mostly easily conducted on-the-fly, in person. To ask people to catch themselves whenever they are making a decision that affects someone who is not in the room is quite challenging and difficult to monitor.

3.2 Time Zone

Time Zone Problems Many informal meetings happen in hallways while team members in Hyderabad are 7,782 miles away, and asleep. Many formal meetings happen in the afternoons, when the other half of the team *should* be asleep. Often, the team in Hyderabad will wake up at 2:30am to participate in a conference call to Redmond. This can lead to a strong sense that distributed teams feel remote in all senses of the word. Turnaround time for electronic communication increases overhead by two or three times compared with communication with someone in a compatible time zone (Boston is for most of the day compatible with Redmond). A problem most easily solved by face to face or phone communication can drag on without resolution due to the inability to communicate synchronously.

Time Zone Solutions Get rid of distributed teams. If you must work with remote teams, encourage people on the team visit the other locations to meet people face to face. Create a buddy system — pair a Redmond person with a Boston or Hyderabad person to li-

aise with and keep each other informed. Encourage more work/life sacrifices from Redmond, e.g. hold meetings early in the morning or late at night to accommodate non-USA schedules.

Working together would be much simpler if the entire team fit into one room. Amazon has a notion that a team cannot be larger than what can be fed by two pizzas. We presume that means they are co-located in Seattle, Washington. Unfortunately, development talent is scarce in the USA, requiring companies to look to other countries for additional help. Additionally, the USA is surrounded by two large oceans on either side, making the American workday schedule impossible to sync up with Europe and Asia.

Creating an exchange program for engineers to visit the other distributed sites is an idea that has been tried with great success. Here is a quote from one of our interviews: "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."

Not only does empathy with the other location and time zone increase, but visiting for longer periods of time creates a personal connection between two remote team members. This connection results in faster turnaround time on emails compared with someone you do not know and leads to an obvious buddy for a buddy system. Visiting for long periods of time is not a scalable solution, however, due to travel costs and family costs. Many people who work at Microsoft are married with children and cannot easily leave their families for a month at a time just to improve relations with team members. Family tension can also result when team members have to wake up at odd hours of the night to meet synchronously with remote teammates.

3.3 Status

Status Problems Waiting for dependent teams to inform you of status causes anxiety. There is often too little communication from the person doing a task to the requester of that task. People find it challenging to keep track of all the things they have to do for others, as well as the things they have asked others to do for them. Expectations of speed, service, and reliability are often uncommunicated, leading to misunderstandings and conflict.

Status Solutions Hold frequent, regularly scheduled status meetings with your dependencies, face to face, if possible. Post status updates online so that everyone can monitor progress. Align your team's priorities and schedules with your dependencies. Tell dependents what place they hold in your priority queue. If another team has you too low down in their priority queue, drop them as a dependency and find another team to fulfill your needs.

When asked during our study, most interviewees reported depending on around 10 people. Most said they sent emails to request tasks to be done, and they check up on that work by sending email. Face to face status updates occurred most often when both people worked in the same building, on the same floor, in the same hallway. If they were in different buildings, the chance of a face to face meeting dropped by half. If dependencies do not overlap in workday hours, turnaround time for status updates often exceeds one and a half workdays. Requiring each team member and his many dependencies to schedule meetings to discuss status would easily cause meeting gridlock. Engineers all report they have too many meetings to go to already, which they feel prevents them from getting their *real* work done.[2] Most status updates are ad hoc as well, making it tedious to formalize and post online, for example, in a bug tracking system.

Team members we spoke to all agreed that aligning the product schedule with one's dependencies can resolve many of the version mismatch problems with concurrently developed software, especially when dependent libraries might be updated midway through a product cycle. If every product was in sync, everyone's highest priority bugs could get resolved in all of the products before shipping. However, it is untenable to ask even dozens of products in a company to all ship at the same time. Like a clocked processor, all teams would work at the pace of the slowest, most complex product. In addition, not every team will agree to work in sync, due to independent business motivations.

All product team members want to help out their customers, especially internal customers. They often do not say no when a request to fix a bug or add a feature comes in. However, they have their own priorities that are not the same as their dependents, and the priority of an external bug or feature can easily be dropped. Without constant vigilance and protest by dependents, their requests could easily never be satisfied. Without attending the other team's meetings, changes in priority for a work item are always surprising. The lack of transparency of one group's processes to an outsider easily leads to anxiety. Teams do not tell others explicitly where they stand in the priority queue, leading to organizational anxiety as well.

4. WHERE ARE THE REAL ANSWERS?

It is obvious that there are no simple answers to coordination problems, even when the problems appear to have simple solutions. Even simple actions can cause unexpected results in complex systems. They can interact badly with individual and group behaviors, especially when those behaviors are part of one's human nature or pre-existing, non-work culture. In addition, the lack of transparency in work processes is leading to anxiety, and an inability to act rationally with full information.

Perhaps the study of coordination should follow a similar path as the study of economics. Originally, researchers assumed that all players were independent, rational beings who acted with complete information. Over time, their theories accounted for groups of individuals (firms) who sometimes behaved irrationally (swayed by emotion) and without complete information (poor communication, hidden processes, and people who manipulate others for their own ends). The attributes considered by these more modern theories likely underlie organization behavior and coordination as well.

Is it possible to create a set of treatments for coordination problems that are independent of the software team that needs them? From experience, we know the teams appreciate seeing data that validates their intuitions about the issues they face, and certainly want to see that data before listening to any researcher who claims to have the answers. We wonder if the researchers who have developed software tools to promote awareness [2, 3, 5, 6, 11, 13], or connect remote people together [7, 9] have reports on the effectiveness of these tools in software teams that were not part of their requirement studies. How would such an intervention be proposed, carried out, measured, monitored, and evaluated?

I am looking forward to this workshop because of the opportunity to speak with other researchers who consider treatment of coordination problems as important as discovering their causes. Perhaps with a little training in psychotherapy, group therapy, and organizational behavior, we could better guide our software team patients to adapt our solutions in ways that most effectively address their needs.

[2]Self-esteem and self-worth issues aside, coordination overhead is often not perceived as contributing to task completion.

5. REFERENCES

[1] Andrew Begel and Nachiappan Nagappan. Global software development: Who does it? In submission.

[2] Jacob T. Biehl, Mary Czerwinski, Greg Smith, and George 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.

[3] Christine A. Halverson, Jason B. Ellis, Catalina Danis, and Wendy 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.

[4] James D. Herbsleb and Audris 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.

[5] Susanne Hupfer, Li-Te Cheng, Steven Ross, and John Patterson. Introducing collaboration into an application development environment. In *Proceedings of CSCW*, pages 21–24, Chicago, IL, 2004. ACM Press.

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

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

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

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

[10] Robert J. Sandusky and Les 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.

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

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

[13] Erik Trainer, Stephen Quirk, Cleidson de Souza, and David 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.