The challenges of staying together while moving fast

2016 IEEE/ACM 38th IEEE International Conference on Software Engineering

The Challenges of Staying Together While Moving Fast: An Exploratory Study

Julia Rubin and Martin Rinard
Massachusetts Institute of Technology, USA
mjulia@csail.mit.edu, rinard@csail.mit.edu

ABSTRACT

We report on the results of an empirical study conducted with 35 experienced software developers from 22 high-tech companies, including Google, Facebook, Microsoft, Intel, and others. The goal of the study was to elicit challenges that these developers face, potential solutions that they envision to these challenges, and research initiatives that they think would deliver useful results.

Challenges identified by the majority of the study participants relate to the collaborative nature of the work: the availability and discoverability of information, communication, collaborative planning and integration with work of others. Almost all participants also addressed the advantages and disadvantages of the current "fast to the market" trend, and the toll it takes on the quality of the software that they are able to deliver and on their professional and personal satisfaction as software engineers.

We describe in depth the identified challenges, supporting our findings with explicit quotes from the study participants. We also put these findings in context of work done by the software engineering community and outline a roadmap for possible future research initiatives.

1. INTRODUCTION

Software development is a central activity in our society. The success of many of the most visible, prominent, and successful organizations is founded on development efforts that produce valuable software or valuable services enabled by that software. The academic software engineering community has a long history of identification of and advocacy for a variety of software development methodologies, goals, and practices, e.g., [6, 4, 48]. Yet, despite decades of research in the field, software development remains an acknowledged difficult and complex undertaking.

Motivated to better understand the problems that modern software developers face, we conducted an exploratory study involving software engineers from prominent, successful high-tech companies including Google, Microsoft, Facebook, Intel, and others. Our study is distinguished by the range of the participants, which come from many companies and many geographical locations. Specifically, we talked with 35 practitioners from 22 companies in four different countries (see Table 1). All of the study participants are experienced software developers, with 12 years of experience on average, and all are actively writing code for production deployment in current software development projects.

The study was organized as a set of semi-structured interviews, which were designed to elicit an articulation of the most important challenges and problems that the engineers face in their current positions. We also asked the study participants to outline potential solutions that they envision and research initiatives that they would find useful.

"Move Fast, Break Things": The interviews revealed that developers are under significant pressure to deliver new functionality and new software products quickly. This pressure comes from their organizations, who in turn perceive themselves as involved in intense competition, with victory going to the organization that moves the fastest.

Many of our participants recognized that this approach works well for their organizations. At the same time, the participants believed that this approach often hampered their ability to deliver quality software. In any case, "move fast, break things" is the reality that establishes the context in which developers must operate. Within this reality, our participants identified interaction and communication as two of the most significant challenges they face in their work.

Interaction and Communication Challenges: Essentially all of our participants are involved in organizationally complex software development activities that require them to work with multiple pieces of software developed by multiple teams both within and outside their organizations. Given the rapid pace of software development and the complexity of the development efforts, the following emerged as the main concerns:

- The need for global, up-to-date, structured and easily accessible information, both internal and external to the organization.
- The ability to exchange information, at an approachable level of abstraction, with multiple stakeholders involved in the development effort.
- The need for effective communication, coordination and planning across teams and geographies.
- The ability to manage the complexity of software that involves multiple components, all changing simultaneously, while controlling interactions and reducing breakages associated with their integration.

Notably, our participants are aware of the current state-of-the-art recommendations in software engineering. Most, if not all, practice agile development with short development iterations of about two to three weeks. Indeed, it is clear that these short iterations and

ICSE '16 May 14-22, 2016, Austin, TX, USA

ACM ISBN 978-1-4503-3900-1/16/05.
DOI: http://dx.doi.org/10.1145/2884781.2884871

Table 1: Demographic Information of the Interviewees

Age	min=23; max=55; mean=37.5; median=39
Total experience (years)	min=1; max=30; mean=11.9; median=12
Gender	male=30; female=5
Education Level	PhD=8; M.Sc./MA/MBA=13; B.Sc/BA=13
Companies	Google=9; Microsoft=4; Facebook=2; Intel=2; EMC=1; VMWare=1; IBM=1; Other=15
Countries	USA=17; Israel=15; Japan=2; UK=1
Company size (employees)	min=10; max=350,000; median=1,500
Team size (employees)	min=1; max=44; mean=11.9; median=10

frequent integration help mitigate challenges related to the lack of synchronization and efficient communication. But, as our study indicates, many challenges still remain.

Ways to Move Forward: We outline a roadmap for possible future research initiatives that could help mitigate the identified challenges. These include global, integrated solutions for information management and knowledge sharing; structured knowledge representations; language paradigms and analysis techniques for identifying semantic code dependencies; pedagogic initiatives; and novel processes and development practices that will allow software engineers to adjust to the rapid speed of the market while maintaining their personal and professional standards. We also note that, to make a difference in practice, software engineering methods and tools have to be conveniently integrated into the current development processes and be accompanied by empirical data assessing the return on investment in their introduction.

We hope the results of our study to be useful for further focusing the efforts of the software engineering research community and grounding them on empirical evidence.

Paper Structure: The remainder of the paper is structured as follows. In Section 2, we describe our research methodology. We discuss the business and organizational culture of the current high-tech field in Section 3. Section 4 describes the identified challenges, while in Section 5 we discuss the challenges in the context of existing work and outline future opportunities. Sections 6 and 7 discuss our results and the related work, respectively. We conclude the paper in Section 8.

2. RESEARCH METHODOLOGY

We now describe our selection of subjects, our approach to data collection and analysis, and threats to the validity of our work.

2.1 Subjects

For this study, we recruited experienced software developers with current, hands-on, development experience. More precisely, our selection criteria were for participants to (a) have at least one year of full-time software development experience and (b) hold a current position that requires to actively write code. For identifying the participants, we initially approached our network of collaborators and colleagues who, in turn, further distributed a call for participation in their organizational and personal networks. We interviewed each participant and stopped recruiting new participants when we did not hear new concerns uncovered in previous interviews but rather repeatedly heard concerns that were already identified.

As the result of this process, we interviewed 35 software developers working at Google, Microsoft, Facebook, Intel, EMC, VMWare, IBM and another 14 companies which the interviewees preferred not to disclose. Prominent domains for these remaining 14 companies include finance, healthcare, and defense. Most participants hold the title of Software Engineer or Senior Software Engineer. A few participants, from smaller startup companies, hold titles such as Director of Engineering.

Table 1 presents Demographic information about the participants. Their ages range between 23 and 55 years, with a mean of 37.5 and a median of 39 years. They have between 1 and 30 years of full-time software development experience, with a mean of 11.9 and a median of 12 years. Only five participants are female. Eight participants hold a PhD degree, 13 hold a Master's level degree and 13 hold a Bachelor's level degree. One participant, with 10 years of experience, was entirely self-taught. The fifth and sixth rows of Table 1 present the distribution of participants by companies and country of employment. The last two rows of the table present statistics about company and team sizes.

2.2 Data Collection

We conducted semi-structured interviews that included a set of open-ended questions and took around 45 minutes (min=25; max=80; mean=48; median=45 minutes). Each interview started with general questions about the participant's background. Then, the interview revolved around three central questions. First, we asked the participants to describe the main challenge they face in their current project and organization. Then, we asked the same question w.r.t. all their previous experience. Finally, we asked them to discuss research initiatives that could be helpful in their work. We followed up with subsequent questions that depended on the interviewees' responses. Our goal was to identify the set of challenges they face and understand the causes and consequences of each challenge.

The interviews were conducted in English. Three interviews were in person and the remaining ones over Skype. All but nine interviewees agreed to be recorded and the collected data was further transcribed almost verbatim, only removing filler words, such as "so" and "you know", and breaking long sentences into shorter ones. The additional nine interviews were summarized during the conversation. We then shared the transcripts with each corresponding interviewee for his or her approval or corrections. We received several minor corrections, primarily involving the names of companies and tools, and applied them to the transcripts.

2.3 Data Analysis

The exploratory nature of our study prompted us to use *open coding* – a technique from grounded theory for deriving theoretical constructs from qualitative analysis [15, 10]. We used open coding to analyze the collected data, i.e., we analyzed the transcripts line by line and detected *concepts* – key ideas contained in data. The 54 identified concepts were organized into 15 categories, and the categories into two themes: collaboration-related challenges and technical and technological challenges.

This paper reports in depth on the first theme: collaboration-related challenges, as these were mentioned by the majority of the participants. Technical and technological challenges were more sparse, i.e., we did not observe any trend common to all participants of the study. Instead, it appears that these issues are more specific to the exact activities that each developer performs. For completeness, we summarize these findings in the appendix of this paper.

All our findings are linked to quotes extracted from the interviews (376 quotes in total), allowing us to ground the results we report and attribute them to the collected data. For quality control, we sent a draft of this paper to all interviewees, asking them to comment on any misinterpretations that might have occurred. We addressed all received comments, mostly related to confidentiality issues, in the final version of the paper.

2.4 Threats to Validity

External validity. As in many other exploratory studies in software engineering, our research is inductive in nature and thus might not generalize beyond the subjects that we studied. Yet, our sample is large enough and diverse enough to give us confidence that it represents central and significant views. We intentionally sampled employees of a diverse range of organizations, small and large, across a broad range of industries. We also intentionally included in the study software developers of different ages and a diverse set of education and professional backgrounds. We believe that these measures helped to mitigate this threat.

Internal validity. We might have misinterpreted participants' answers or misidentified concept and categories, introducing researcher bias to the analysis. We attempted to mitigate this threat by sharing both the "raw" data collected during the interviews and the resulting report (this paper) with the participants of the study for their validation. We thus believe our analysis is solid and reliable.

3. "MOVE FAST, BREAK THINGS" TREND

IF YOU ARE NOT MOVING AT THE SPEED OF THE MARKETPLACE YOU'RE ALREADY DEAD – YOU JUST HAVEN'T STOPPED BREATHING YET. — Jack Welch.

Out of 35 interviewees, 25 explicitly mentioned significant pressure to deliver fast to the market, even at the expense of reduced quality and accumulated technical debt. "For many of these 'Silicon Valley tech companies', there is a lot of pressure to demonstrate that your work is immediately visible and 'impactful': important to the company or the industry." [1] Facebook's motto until April 2014 – "move fast, break things" – appears to be the norm in industry, according to participants of the study. "We are so focused on delivering right now that there is a couple of things that suffer in the meantime. Quality is something that suffers. As one of my colleagues said, we are first and foremost a product company and not a technology company, which means that sometimes we sacrifice quality, because we are trying to get our product out, as fast as possible." "This is all about the whole idea of MVP – a minimal viable product. That is what many companies, in particular start-ups, do: trying to understand what is the least level of investment and time commitment that you can have in order to release a feature. The idea is that you want to release something that is at a functional level, but not necessarily has all the best features and the most spectacular version of them. Instead, you add on to your product over time."

We did not explicitly elicit participants' opinion on that trend. Yet, several, from both start-ups and large companies, believed it is a necessity nowadays, as it allows a company to establish itself as a market leader and achieve a competitive advantage: "My experience is that any time you try to aim for something perfect, you miss what you can really gain." "You just need to live with this reality. Unless we can stop the market; tell other companies: let's stop for a couple of years, until we get everything done."

Others believed that the trend induces development standards that are harmful in the long run: "There is an impression that getting something out of the door is more important than getting a good thing out of the door. Which is OK, if you happen to be building a website that nobody's life depends on. But it is unfortunate, because it is a style of engineering that you become socialized to. And you cannot conceive how to engineer any other way, without extensive retraining. So many people have been socialized to just cram something out into the world, with a minimal amount of consideration. They believe that you can release it and then fix it after the fact. Regaining the skills of doing something in a way that it has controlled failures, it is field serviceable, it is well-understood by the user and the engineer, is a set of skills that will take a long time to relearn."

How to move forward? In reality, software developers have little influence on the business strategy of organizations. Without a serious business or financial incentive, the pace of mainstream software development is unlikely to change in the near future, because companies "still get what they need": "They believe we are doing well enough. The engineers will manage and they always do, even under time pressure and with insufficient resources."

Yet, this reality has a set of consequences that the developers perceived as negative and that disturb their ability to comply with the companies' demand and deliver fast to the market. It became apparent from our study that most, if not all, our participants are aware of state-of-the-art recommendations in software engineering. Also, most practice agile development, e.g., Scrum [48], with frequent, two to three week development cycles. However, these practices are unable to address all challenges and there are gaps that still have to be addressed: more than 20 participants mentioned challenges related to the collaborative nature of work, i.e., efficient communication, synchronization, integration and information exchange, as discussed in Section 4. "The best thing to do would be to look at how to make sure that the products are done very quickly, they flow out very fast, the development is very quick, but you also maintain good inter-team communication and make sure everyone is always on the same page. It is possible to release things quickly, and it is also possible to make sure your teams communicate well. But I think where most companies fail is doing both at the same time without hindering the development cycles. So I think finding some sort of planning model or design cycle where the speed is still there, but you are having a lot more interaction with different teams in the company, is something that would be very valuable."

This situation calls for methods, tools, techniques and processes that will help developers to thrive in a collaborative environment and to increase quality and productivity, while mitigating the professional and personal toll that the current software development process takes. Towards this end, in what follows, we describe in depth the identified challenges (Section 4) and outline some opportunities for addressing them (Section 5).

4. MAIN FINDINGS: COLLABORATION-RELATED CHALLENGES

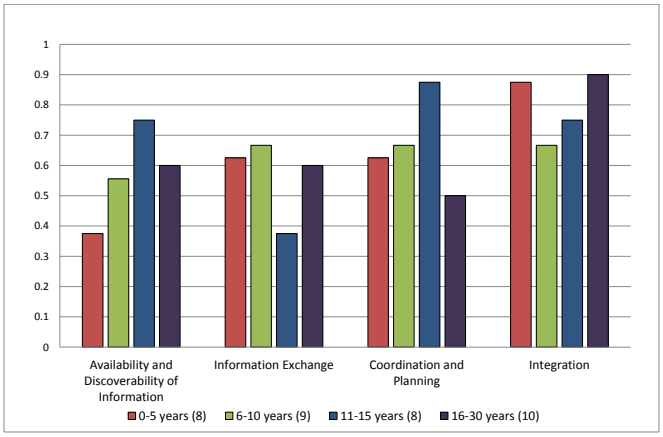
The vast majority of participants mentioned issues related to the collaborative nature of the work: availability and discoverability of documented knowledge, communication within and between teams, coordination and integration with work produced by others. Table 2 presents the number of interviewees that raised issues in each of the categories. Figure 1 further breaks the information down by years of experience and education. Interestingly, we did not find any meaningful correlation between the demographics of the interviewees and the challenges that they raised. That is, the challenges described in this paper were raised by engineers of various ages, professional backgrounds, educations, etc.

4.1 Availability and Discoverability of Information

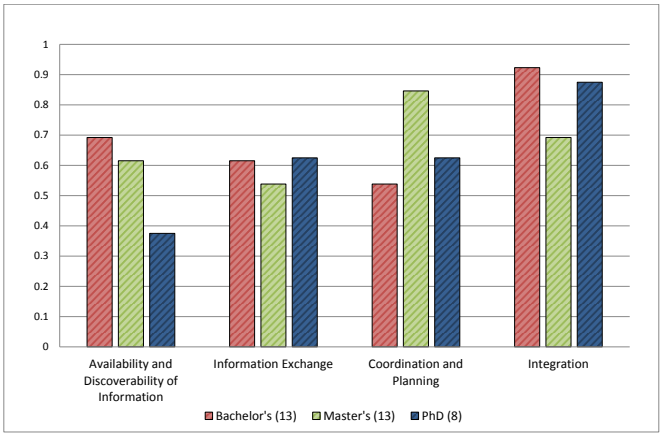
4.1.1 Documentation

Developers often lack sufficient time to invest in documentation. "It is just low priority, you need to deliver as soon as possible".

[1] This paper often presents direct quotes from the study participants, as in the case above. The quotes always appear in "this style."



(a) By Experience.



(b) By Education.

Figure 1: Fraction of Interviewees that Raised Collaboration-Related Challenges.

Table 2: Number of Interviewees that Raised Collaboration-Related Challenges

Category	#
Availability and Discoverability of Information	20
Information Exchange	20
Coordination and Planning	23
Integration	28

However, several study participants find this lack of documentation problematic: "Our project uses a lot of pieces which are not very well documented. The lack of documentation is a big challenge, and in combination with the complexity of the system, it is hard to understand what the different pieces do. You have to dig deep into the code to understand what is going on."

Studying code only is, however, often insufficient: "There are several undocumented invariants in the system. Some are manifested as assertions in code. When I make a change, I want to understand whether an invariant is an inherent, essential part of the design or whether it can be modified if needed." "You can see the check-ins, you can see the code of the whole program, where people decided to do things in a certain way, but there is no 'why'. You only see the results. There is no rationale."

To overcome this problem, "anytime you really need a question answered, you have to figure who built it and go ask that person. Of course, you will receive a lot of information, but it is also a waste of time. And if that person has to answer questions, they really need time." When the person responsible for that undocumented code leaves the organization, valuable information is lost: "The code is so old that no one really remembers exactly how it works anymore." "Just last week I answered a question from my previous project. One person had a question and nobody there knew the answer. And that is a relatively new project; that decision was made around a year ago. Luckily, I still remember and I am still in the organization. But if not, you might revert the decision and get burned over the same problem again."

Even when documentation is available, it is often obsolete: "You never know if it is updated, because things change so rapidly that a feature could be working in one way, and changes happen, and no one is going to update that documentation." "Even if people do write why they do something, when they change the design, nobody goes back and changes the design documents."

In some cases, the documentation is incomprehensible: "Our projects last for 10 or 15 years. Documents are available, but they are hard to read. They are written 15 years ago and they use a kind of vocabulary that you do not understand completely, because it is vocabulary of some specific project." "People who are writing code forget about what is obvious to them, because they have just been thinking about it and writing that code for so long. Things that are so obvious for the people who have written the code that they do not even bear mentioning, might not be obvious at all for someone who reads the code later."

4.1.2 Information scattering

Numerous participants mentioned issues related to information scattering, i.e., lack of global, cumulative and up-to-date information sources for design and code changes: "We are trying to develop summaries, to keep track of what is going on, etc. But for now we do not have a very structured way for doing that, and no central repository for keeping the summaries." "In the rapid pace of software development, keeping a huge repository of documents is not something that answers it."

Many changes are described in commit messages of configuration management systems: "People describe their changes in check-in messages, but in a check-in you cannot draw a figure. Also, there are many check-ins for a design change. So, every check-in has some small descriptions, but one does not have any centralized place to see why you did all these changes."

Instead of looking for design changes, developers then have to analyze bug fixes: "You have to remember that somebody made a change to fix some bug, and search for the bug commit message. Maybe there the developer wrote why they decided to change the design. But to find it, you have to go roundabout. You cannot say: why does this thing work in this way?"

Another participant also noted that commit messages are too granular and are usually oriented towards the code that has changed rather than the "high-level human interface" which provides operational information: "I am rather looking for something like 'Hey, I have changed the database schema. You might have to delete your local database and recreate it.'" Such information is rather distributed via e-mail, which is an additional contributor to information scattering: "Most of the organizations that I have seen so far rely on email too much. It is not searchable, it is not accessible to everybody, and once I leave the organization, all my email gets deleted. It is not a good medium."

Yet another challenge is a variety of versions, each requiring its own documentation: "If you want to understand the design of the system, it often happens that one feature is documented in version 18, but it relies on a feature that is documented in version 15. So you

need to go back and read the design of version 15. But the design document of version 15 relates to the system in version 15, which is different from what you have in version 18."

Participants noticed that, similar to internal information, external information is often scattered in many different places, such as tutorials, APIs, and FAQ pages: "You do not have a full picture on what is going on. You need to search in different places to figure that out." Some external, customer-facing, information is hard to find: "Because we work with healthcare data, there are limitations, primarily around how we use the software to process the data. Amazon would tell you that 'yes, you can use the Elastic MapReduce on healthcare data'. But the documentation and the patterns that you can find online are not sufficient."

4.1.3 Finding experts and know-how

Finding experts and expert opinion on a particular problem is also challenging. One participant, working in a very large organization, mentioned that he knows for sure that there are people within the organization that are experts in a particular topic, but it is unclear how to find them: "We develop a driver for AIX: an IBM operating system. IBM is not very open about the documentation for AIX and there is not much information on the Internet. We know that there are people in our organization that develop for AIX, we just cannot find them. And if you do find somebody, then you want to find somebody with an experience that is relevant to what you are trying to develop. If you work on a physical driver of storage, you do not need somebody that developed a UI for AIX."

Much information is nowadays available online: "Right now, when people are looking for an answer to a technical problem, they search on Google, and a good resource is something like StackOverflow, which can tell you how to solve a code problem." Yet, this source of information has some deficiencies: "Searching and reading there takes a lot of time. You need to open each thread, read it and understand if the solution fits or not. Then you need to try it and see if it works. And sometimes solutions are simply not there."

The information is unstructured and difficult to navigate: "If you want to know 'I have data with these characteristics, or a system that has these needs, what third party tool should I use? Who else has done what I am looking to do?', the solutions here are very bad, it is very hard to find that information." In addition, particular problems, e.g., related to equipment tested under NDA, cannot be freely shared online. Thus, information about failures in such cases is often unavailable.

Finding best practices and advice from academic literature is another problem that was brought up by a number of participants: "[I would like] to find approaches that are going to work, given a specific set of criteria. It would be great if someone could say 'One, two, three, here. These are the approaches that work for you'." "Architecture, optimal splitting of your code base, is probably a very old problem. How do I find the optimal way of doing that?"

4.2 Information Exchange

Another challenge, mentioned by 20 participants, is communication with various stakeholders involved in the project and efficient information exchange between multiple parties.

4.2.1 Interpreting information

Different people might have a different interpretation for the same piece of information: "People have different technical backgrounds, as well as different constraints, and all these result in different perspectives on both the problem as well as on the solution." "You can ask someone to do something, you can think that you have written the best spec that you have ever written in your life and you will still get a different piece of code. If you ask for a feature and there are three real core pieces to said feature, each person will weigh the value of each part of that feature differently. And then, when they implement it, they may optimize for one over the other, not consider the general case well enough or just misunderstand what the general case likely is." Participants mentioned that such misinterpretations can cause them to lose time re-implementing features again and again.

4.2.2 Speaking lingua franca: multidisciplinary teams

Information exchange becomes even more complicated when team members have different technical backgrounds, expertise and roles in the project: "Our team is not just comprised of software engineers; we actually have a lot of medical people who are working at the company. The ability to do knowledge transfer effectively is, I think, our hardest challenge."

In some cases, participants envisioned solutions that would allow to translate the information from a representation that is convenient for one stakeholder to that of another, but such solutions cannot work in the general case: "Even the non-technical people are on the spectrum of technicality, so I can imagine a biostatistician coding in R, and then a compiler translates R into Python. I probably would not like it and I bet the code would not be very good, but I can imagine a solution like that existing. If you also have a medical person, a genuinely medical person like a medical doctor, who does not write code at all, then it is much harder to think of a technical solution."

Related is the problem of communication between developers and product managers, and between product managers and customers: "I need to work with researchers, engineers and product managers, and act as the person that understands all the technical constraints and is able to communicate them to all the parties involved. The harder part here is to be able to communicate problems and solutions between different levels of technical abstractions. Different backgrounds lead to different views."

Several participants mentioned that product and program managers do not necessarily get to the bottom of all technical considerations: "A product manager defines it at a very high level. Then issues come up during the development time and we need to change the specification. The idea that is in the product manager's head might appear different to the developer." "People who specify the requirements do not know what they are asking for. Sometimes, I think, they just do not speak the right language." As a consequence, solutions are often underspecified, or, even worse, are specified incorrectly: "They think things can work one way, but the client's network configuration makes that impossible, so they do not understand the requirements." "Even when we think we have got it so clear, and that we know what we need to do, half-way through we change course on something."

4.2.3 Politics and bureaucracy

Several participants mentioned that internal politics often take time and energy. "You constantly have to discuss and convince other people, until everybody agrees and is happy with the decision. It feels like you have a responsibility but no organizational power to act on it. That takes energy." "At this stage, the organization is too political. It takes a lot of effort to push an idea forward. It takes a lot of energy to argue with people just for the sake of argument. The environment is very competitive, and some people speak out their opinions just in order to speak out."

Politics and miscommunications can cause developers to lose valuable time: "I have had to redo the same feature multiple times because of disagreements between management. The feature itself

was not complicated, but getting multiple parties to agree on a specific implementation was a challenge." "One thing I have seen a lot of is that there are some people who are very proud of the code that they write. Even if someone objects to the way they have written a certain piece of code, their pride will stop them from being convinced to do it differently."

Some participants, especially from smaller start-up companies, noted that building good interpersonal relationships and healthy communication channels often gets a low priority: "Start-ups do that at the expense of the long-term stability of the company with respect to inter-team relationships and communication. That stuff just goes ignored in the start-up style, because the product is more important. But if you leave that kind of relationship too long, that will eventually cause major problems down the line. Just making sure that everyone is in sync, everyone is on the same page, everyone is working well together and everything is in harmony is important."

4.3 Coordination and Planning

As much as 23 participants mentioned the challenge of coordination with local and remote colleagues.

4.3.1 Coordination between teams

Lack of efficient communication and coordination between different development teams is a challenge that was brought up multiple times: "When collaborating on your own team, there is very good communication, we sit together, we have a lot of social interaction. There is encouragement to brainstorm a lot. Every team is very much united. However, I realized that we really do not have any good structure for collaboration between teams. Whenever we had to collaborate with another team, we faced a lot of resistance, frustration, conflicts that had to be escalated and resolved by higher management, etc."

Isolation and lack of information exchange between different development teams tend to affect planning, delivery quality and speed in a negative manner: "The biggest challenges that I have seen so far is that different development teams have very different ways of doing things. It is very complicated to actually work well with the teams because they all do things in different ways. And then once you have done something, if there has not been any real communication between teams in advance, you could easily have to redo the whole thing, because a different team disagrees with how it has been implemented." "The product you are going to build works as part of a system. There is a big barrier to overcome here because you need that domain knowledge to be successful. So you could think that you are doing something really, really well, but you are actually making it really difficult for the teams. And they could be making it difficult for you because they are not considering the technical difficulties they are introducing to your part."

Such problems appear to be more prominent in smaller companies that recently experienced growth and thus did not adjust their practices to the newer realities. Several participants noted that these challenges can probably be attributed to the "growth pain" of the companies: "We are not at a point anymore where we are working in silos. We achieved what we could in that way, and now a lot of our projects are interdisciplinary, so to speak, between groups. We need to shift our organizational structure, so that we have someone who will negotiate with multiple teams and will make sure that each team is aware of where their involvement is required."

The need is for technical, rather than high-level, managerial coordination between teams: "It just has to be a sort of liaison between the teams. That is, a person whose role is to communicate effectively. The person does not need to be from the management but rather somebody technical who knows what is going on in that process. That way, there is at least some knowledge about what is happening [in other teams]." "Because program managers are not developers, they do not know exactly what [a specific problem] means. It really ends up being the developers that need to negotiate that with each other. And obviously, as developers, we have other high priority issues to deal with in our own team."

4.3.2 Collaborative scheduling

Teams do not get any time allocation for helping others and contributing to their tasks: "When projects are being planned, they do not take into account that you will need that other team's time. That other team also has their own roadmap for the year. They do not have time to help you. And you are blocked because you need them. There is no way that you can make progress, otherwise. Even if you do the work, our code bases are pretty much divided according to the teams. So it really ends up being that you need to write code in some other team's collection. And they still need to do the reviews. They still need to help you with the architecture, because you need to do something in their code base. That creates a lot of conflicts. As a result, anytime you have to collaborate with another team, you have to go and beg them for their time. Their time is not accounted for in the planning of the project."

From the other end, interrupts and context switches are some of the major pain points that came up often: "When you are switching between tasks, you lose a lot of energy. And it also costs energy to get back to the same task later on. Context switches are time and energy consuming." Interrupts do not come only from demands of other teams, but also from supporting the released products: "Things that are coming from support burden, be it feature requests or bug reports. These can take a lot of energy, to the point that working on any new task or new products becomes difficult." "Better specialization and separation of tasks would help to improve productivity."

4.3.3 Coordination across locations

Coordination problems become even more acute when teams are distributed across multiple locations: "Time zone differences are difficult. Coordinating people that are in different time zones is a nightmare." "Time differences, language differences, and the lack of physical access to people we work with make us less productive."

There are two main sources of this frustration. The first one is coordination of work tasks: "Our day is their night and the other way around. To get stuff done you need to be available in late hours, to talk to people verbally and not via email, as when you communicate over written media the communication is always slower, sometimes ambiguous." "I check my code in, go home, and get their feedback only on the following day. I reply, they see it after I go home and so on. If I stay up until midnight and work with them online, it works well. Otherwise, the cycle is very long." "If all the parties were assembled in the room together at the time of problem formulation, I believe it would mitigate most of iteration cycles I underwent. The distribution of engineers between India, Singapore and California that may all be working on the same project, make it challenging to keep synchronized. The manager of the project I was on was located in California, I have never met him. One of the other engineers that would conduct code reviews was located in India." Interestingly, these challenges were brought up by the interviewees from the US, Israel and Japan, so the problem is not specific to any particular geographic location.

The second source of frustration is more personal. With all the available technology, such as email, chats and video conferences, several participants still feel that personal connection is important. "A lot builds up on personal relationships. It might sound funny, but that is true. If you know the person, he will do things faster for you,

because he feels uncomfortable to ignore you." "You need to know people personally, in order to get something out of that."

One participant addressed the lack of investment in team building activities: "If everybody is committed to the same goal and everybody knows each other well, productivity can go up and the 24h work can happen. It is a psychological issue. Today, I can say that our team does not really know people in Seattle well. Our communication is problem-based. That is, we contact them if there is a problem and there is no other day-to-day communication. As the results, the work is not flowing that well. It is always easier to approach a person that you know well than some remote person that you barely know."

4.4 Integration

The most mentioned (by 28 out of 35 participants) was the challenge of integration: building a large system while controlling interactions between all its different parts. Code rarely exists in isolation, but rather integrates with a set of internal and external components and frameworks. Dependence on artifacts developed by other teams can introduce unwanted behaviors or even break the system when the code of other teams changes in an unexpected manner. To a large extent, it appears that the limited communication, coordination and information exchange between the teams, as discussed above, could be one of the major contributors to this problem.

Teams also break each other's code because systems are complex, yet developers lack sufficient time to invest in designing their solution and evaluating different alternatives: "In a perfect world, where I had an infinite amount of time, if I would create a piece of software, I would first create a design document and have it reviewed. From there, I would probably build a prototype or a proof of concept, to show it is going to work. And then, once we got it working and things are the way they are supposed to, rebuild it all but meant for production." Yet, another participant mentioned that "you do not get as much time to become comfortable with what you have to do before you go ahead and build the actual product. The typical cycle for design to implementation here is about two weeks. So you really do not get that much time to spike on what you need to do. You just have to jump straight into it."

APIs between different parts of the system are, in some cases, not defined well: "It is easy to say but hard to implement. Good interfaces are hard to implement because code is complex, the architecture is complex. When you design the APIs, you do not always think about all related stuff. The second reason is that many time we are in a hurry, want to do things fast. So we define the APIs quickly and say 'let's start working, see later if we need to make changes'."

Frequent changes in APIs cause developers to lose time: "We get the information, we get the update, but we have to modify our project as well, which is not that great." Yet, changes in the APIs are the easiest to detect and, while several participants mentioned that such changes can slow them down, the majority of problems relate to "semantic" changes: "We are aware of the dependencies. It is rarely a syntactic problem. There are always changes where their effect is not predicted by the changer, not predicted by the product leader, but may affect some feature in a specific state. There is always some implicit behavior that someone relies on, and others may change it." "You are doing something, and somebody on the other side of the system does something completely different. You get a butterfly effect – as in the chaos theory, when a butterfly flaps its wings on the other side of the project it can produce a hurricane on another." Good solutions for identifying semantic dependencies between components are missing in practice.

The integration problem is even more challenging for developers working in safety-critical, financial and other sensitive domains, who spend extra effort on analyzing the runtime behavior of all third-party components, so that their behavior can be predicted and modeled: "There are things that we ship with, that we did not build, that we do not strictly control. We have external systems and we have other teams, and we consider their systems to be similar in terms of external systems: they are external to us. But we limit their change and we have probes around them so that we can see if they are behaving in odd ways. That is, every time that we introduce a new component that we do not control into the system, we build hosted probes around it. We run it in a limited capacity to get some sense of what its "normal" is. It is sort of an expanding set of interlocking circles, where you have domain knowledge and you have background knowledge, and you have these interpersonal relationships. And the overlap of all of that determines how well you can reason through or construct experiments for determining the behavior that violates your implicit model of how the system should be behaving. But where possible, we prefer to build things ourselves."

5. A ROADMAP OF EXISTING RESEARCH AND FUTURE OPPORTUNITIES

In this section, we relate the identified challenges to existing work. We then contrast this work to solutions proposed by the study participants and derive recommendations for future research directions. We invite the community to join in analyzing the results of this study, exploring the applicability of existing solutions, and proposing novel approaches that mitigate the identified challenges.

5.1 Existing Research

A line of work on facilitating information discoverability includes work by Moreno at al., as well as Martie at al., who propose solutions for improving code search capabilities [36, 34]. There are also approaches that aim to reduce information overhead in issue-tracking systems [2, 8] and in change management systems [41]. Sorbo et al. [49] consider approaches to classify the content of development emails according to their purpose.

Mockus and Herbsleb propose a solution for facilitating expert identification [35] by using data from change management systems to locate people with desired expertise. Recommender systems also attempt to tackle the problem of expert identification [45].

Several existing works propose approaches for better code management, which include summarizing and explaining code [43], API usage [36, 42, 39], managing integration conflicts [38], and commits in version control systems [27, 1, 32, 37].

The global software engineering community has considered topics related to increasing developer awareness when working on large code bases [12, 16, 21, 25, 50, 17, 47]. For example, Calefato and Lanubile [7] present a tool that augments Application Lifecycle Management platforms with social awareness. The tool facilitates the establishment of interpersonal connections by allowing the developers to disclose personal interests and contextual information.

Some solutions to geographic, temporal, cultural, and linguistic distance suggested in the literature are non-technical and include frequent site visits, assignment of dedicated people responsible for multi-site and multi-stakeholder collaboration, and suggestions to capture implicit knowledge and make it explicit [40, 30].

Agile methods [4] encourage teamwork, frequent inspection and adaptation, rapid delivery and a business approach that aligns development with customer needs and company goals. Extreme programming [3] and Scrum [48] are probably the most widely known and used methodologies based on the agile principles. Later works also look at how agile practices are used in large and globally-distributed software projects [24].

5.2 Future Opportunities

Processes. Our data suggests that developers are largely aware of the current, state-of-the-art development processes recommended by the software engineering community. One participant shared a successful synchronization experience, which was inspired by Scrum and was practiced in his previous company: "We had to produce a demo for each developed feature and show it to the product managers. The product managers then said whether it is working as expected, or whether there are problems that they can spot in the solution. We had to do an actual formal demo for each feature. There was a dedicated day where all completed features were demonstrated. The developers received feedback and that was very helpful. It helped to reduce the gap in understanding and we were able to catch many issues with that process." Yet, despite this positive experience, a similar process is not practiced in his current company.

Future studies are needed to explore why practices that are shown to be successful in some organizations are not employed by others. Moreover, several participants of our study mentioned that companies often employ customized versions of known practices, adapting them to the need of a particular organization. While such adaptations are indeed encouraged by agile methods, it may be time to conduct more detailed studies aiming at identifying characteristics of organizations that use certain practices and provide specific, tailored recommendations to organizations with different characteristics.

There is also a need for development processes that put more focus on psychological aspects of interaction and communication between people: "The organization should engineer the situation in a way where you feel as you should communicate with people, rather than you are being forced to." More research on optimal development processes that take both development speed and human factors into account is needed.

Knowledge management. Solutions that automatically produce human readable, concise documents are needed in practice. Such solutions should be able to define and adapt the presentation style to the background and communication pattern of each reader.

In addition, there is a need for a central repository that captures the most up-to-date information for an organization and/or a project. Unlike several existing works that promote information gathering from one or only a few sources, such a repository should integrate information from all available sources, including code, commits, emails, available documentation, and bug reports. The repository should be able to present the information to stakeholders according to their format preferences. It should also be integrated with tools and development environments, making information collection automated. Information inconsistencies and conflicts, if they arise, should be reported to the user, at the corresponding abstraction level.

As suggested by one of the participants of the study, devising a structured way to describe problems and their solutions in crowdsourced repositories such as StackOverflow would increase their usefulness and reduce search time: "It would be good to create a format of a problem description / solution, so that people can easily find what they need." Such structured repositories would be beneficial both internally, within an organization, and externally, for experts across different organizations. The knowledge should be cataloged and sliced by categories, e.g., technical solutions, architectural patterns, tool availability: "It has to be very convenient to use. Some problems are at a very high conceptual level, and some are very technical, with very technical solutions. It is difficult to merge all that into the same base framework."

Integrating information sharing capabilities into development environments was, again, perceived as a useful feature that is missing in practice: "Ideally, my development environment should be integrated with the community. If I obtain an error, it could automatically search for similar issues in, say, StackOverflow, because my development environment already knows the exact context of the problem: the type and version of the development environment, the error message, the location of the error, etc. It should also allow me to automatically publish the solution when I find it. That would save time and increase the availability and accessibility of the solutions."

Integration and integrability. Providing technical solutions that minimize the "butterfly effect", i.e., surprising behaviors during integration, appears to be a useful research direction. Effective modularization and componentization can help mitigate the problem, but do not eliminate it completely. Providing increased visibility into and/or guarantees regarding interactions between different parts of the system (by, for example, program analyses potentially augmented with formal specifications) is one potential direction.

There is also a need to devise recommendations for an efficient logical division between the teams, as well as a division between their code bases, that respects social and geographical constraints and allows each team to "actually go and concentrate on their work for a particular period of time without needing to coordinate constantly with another team."

Education. Some knowledge gaps pointed out by the participants of the study could be addressed by software engineering educators. These gaps mainly involve testing, debugging and the collaborative work experience: "I think that there are things that we can do in the area of software engineering education. Students who come out of school do not have a lot of good experience working on testing. They do not have a good experience working on cross-functional teams or on projects that require a lot of people to work together and coordinate deadlines. Or documentation: students do not know how to write a design doc, they don't know how to maintain code or read other people's code, they have never done that." "Teaching people how to debug software that they have written and how to make error signaling methods, like logs and alarms and such, and monitoring, that are meaningful. Building things into your program that make it easier to figure out when something is wrong."

Empirical evidence. Several participants highlighted the lack of empirical data which justifies (or refutes) investment in software engineering processes and tools, in terms of business revenue. Such empirical evidence can help to steer industry into a more rigid and "scientific" approach to software development: "You can't really split actual causes from just circumstantial evidence. What I would like to have is more data. I have no idea how that can be done. How do you even compare a team and say, 'This team is more productive than that one?' I don't know. But what I would like to have is more data. If there are good metrics for deciding how many tests should I have? Is it generally better to release daily? Weekly? And why? Even just empirical data, but if it has a strong statistical significance, it's great. I want to be able to advocate for something based on data. I do not want to argue with people based on feeling."

The participants also noted that productivity is often hard to measure and quantify in terms of revenue: "The bottom line is that the company still gets the results they need. They might get them not as fast as it would happen otherwise, but I guess it is sufficient." "That is how companies operate. They do not see far enough to realize that they could invest some resources and, as the result, make the developers more productive and get more out of them." As a result, productivity-improvement solutions are difficult to introduce: "If there was research that showed how good tools can improve productivity you could influence executives to focus on that more."

A concluding remark. We noticed that convenience and availability are the best incentives for developers to adopt new engineering

practices or methodologies: “Discipline does not scale. The way to achieve robustness is to shape your processes in a way that it is hard to break things. Discipline cannot be expected from people for long. I think there should be a technological change that would also drive process changes.” We conclude that, to make a difference in practice, software engineering methods and tools have to be conveniently integrated into the current development processes. Moreover, the introduced approaches have to be accompanied by empirical data measuring the return on investment induced by their adoption.

6. DISCUSSION

The participants in this study work for some of the most successful software development organizations in the world. While they are clearly under significant time pressure and often feel stressed and frustrated with various aspects of their jobs, the fact remains that this approach has enabled their organizations to become extraordinarily successful. Given their success with this approach, what would motivate these organizations to change?

We believe that a major change can occur only when current development practices stop paying off for the organizations. One such potential change catalyst would be consumer refusal to accept the flawed, complex, or reduced quality software that these methods can deliver. One of our participants identified a market, in the software-driven agriculture domain, in which this is already happening: “Agricultural technology is really fascinating because modern American farmers – there are very few of them and they are mostly supported by machines. Increasingly, farmers are buying tractors from the 1970s. And they are buying tractors from the 1970s not because the 1970s were some sort of golden age of tractor development. But it was the last era in which computers did not sit extensively inside of the tractors. Say the farmer buys a brand new tractor with an onboard computer device, and it works amazing: it can detect exactly what parts of the field needs water, it is an incredible robot. But as soon as it breaks, their farm is dead. You will have to wait for someone to fly out from the city, to fix the computer inside of the robot tractor. And farmers do not appreciate that. But it is possible to build a robot tractor that does not have these faults. It is possible to do all the preliminary work that you need.” In the software domain, consumers might, similarly, opt for companies that provide simpler yet more reliable software as a deliberate strategy.

Security vulnerabilities and privacy violations comprise another potential change catalyst. Our participants were not happy with the quality of the software that the “move fast, break things” approach enabled them to produce. To the extent that security vulnerabilities become a serious enough concern, a need for more secure software could motivate companies to place a higher priority on more principled development practices that make it possible to avoid such vulnerabilities, or at least to minimize their frequency. The visibility and negative publicity of recent security breaches involving, for example, Ashley Madison [54], Sony [56] and the United States Government [55], highlights how this issue is becoming increasingly important.

One of our participants identified a specific project that was unsuccessful due to negative public perception around the area of privacy after significant engineering effort by a major company: “Being more careful upfront and imagining what are the perceptions or misunderstandings that can arise, and then trying to decide if those trade-offs are worthwhile [would be better]. It felt for a while that the push was always to do whatever is going to be the most viral, whatever would help the product to grow the fastest, and not necessarily careful enough about the risks.” Losing money on unsuccessful projects may eventually motivate organizations to take a more principled approach to software development.

Insurance for losses caused by data breaches, security vulnerabilities, and other problems associated with poor software quality has only recently started to become available. Experience in other industries shows that underwriters can force significant changes in industry practice by requiring companies to meet certain standards as a precondition for issuing coverage [52]. It is therefore possible to envision scenarios in which underwriters force companies to move away from their current "move fast, break things" approach to software development.

Until such changes happen, we believe that the most productive immediate direction for the software engineering research community revolves around helping developers to continue to move fast (or even faster) while breaking less, rather than attempting to convince organizations to slow down and produce better quality software. Identification of practices that enable companies to proceed at a rapid software development pace and maintain a high level of quality is needed. Given the extent to which development in these organizations involves extensive interactions between multiple teams and rapidly changing pieces of software, we would expect such practices to focus on helping teams work together better in a complex, rapidly evolving ecosystem.

7. RELATED WORK

Below, we outline empirical studies and agenda-setting papers that discuss communication-related challenges. Unlike many earlier works that focus on a particular development practice or lifecycle stage, our study had an open nature; the fact that our participants have chosen to highlight communication-related challenges emphasizes their importance in practice. Our work thus provides further validation to some of the earlier findings and ideas in this area. It also identifies additional challenges not covered by previous work and proposes possible future research directions.

Organizational factors. Lavallée and Robillard [29] investigate the relationship between organizational factors and the quality of the produced software. The study is based on ten months of observation, during mandatory weekly status meetings, in a large telecommunications company. The authors identified ten organizational factors, such as internal and external dependencies, work under pressure and human resource planning, that negatively affect software quality.

Hannay and Benestad [18] study productivity threats in agile development projects by conducting 13 interviews with members of 11 different projects within a single organization. The authors identify ten problem areas, including the presence of excessive dependencies within a system

Mark et al. [33] discuss the cost of interrupted work. The authors report that, surprisingly, people completed interrupted tasks in less time with no difference in quality. That is, people compensate for interruptions by working faster, but at the price of experiencing more stress, higher frustration, time pressure, and effort.

Organizational structure. Herbsleb [20] lists several collaboration-related challenges in his research agenda for global, distributed and multi-site development organizations. Among the identified challenges are the need for tools that support better communication via virtual co-location. Other authors document collaboration and communication challenges in global development teams, showing that organizational culture, structure, and support has an effect on the quality of the produced software [11].

Lack of alignment between the structure of a software system and the social boundaries of the development organization (cf. Conway’s law [9]) is also linked by earlier works to problems of interaction, collaboration and ultimately quality in software projects [22, 28].

Knowledge management. The knowledge management community documents several problems related to availability and discoverability of information [46, 13]. In particular, these works note that expert identification is a common problem in knowledge management.

Robillard et al. [44] conduct a field study of API learning obstacles, identifying the lack of learning resources, such as documentation, as the most severe one.

Hansen [19] reports on how "weak ties" enable knowledge sharing between different organizational units, based on a network study of 120 new-product development projects. Results indicate that weak interunit ties seem to impede the transfer of complex knowledge.

Storey et al. [51] examine the role of social media in software engineering. Their results indicate that developers value social media, but traditional channels, e.g., face-to-face communication, are still considered crucial.

Whitehead [53] presents a list of goals and directions for improving collaboration in software engineering. These include tight integration between web and desktop development environments, broader participation of end users in the development process, capturing argumentation surrounding design rationale and use of massively multiplayer online game technology as a collaboration medium.

Education. Begel and Simon [5] report on an in-situ case study that monitored software developers in their first six months at Microsoft. They discuss possible changes for the onboarding program and university curricula, e.g., familiarizing students with working on an existing large codebase in a collaborative manner, rather than working on greenfield projects.

Hewner and Guzdial [23] investigate what game developers look for in new graduates. The results show that game companies are looking for programmers who can not only solve algorithmically challenging problems, but can also write and debug code in an efficient manner and are able to work with others. Lethbridge [31] reports on a survey on what knowledge is important for a software professional. Topics with the largest knowledge gap include negotiation, leadership, management, ethics and professionalism and requirements gathering and analysis.

Relationship between industry and academia. Several works [26, 14] explore means and information needs for successful transfer of technology from academia to industry. Among the findings are the need for information regarding the impact of technologies on product quality, cost, development time, and the preferred sources of information, such as colleagues, textbooks, and industry workshops.

8. CONCLUSION

Given the central role that software plays in virtually every aspect of our society, software development is now a critical engineering activity. Our study indicates that developers in the most successful software development organizations operate in a complex, fast-paced environment that prioritizes development speed over software quality and presents developers with multiple interaction, communication, and information challenges. The "move fast, break things" trend has delivered results for the organizations and therefore shapes the reality in which developers operate. Productive research directions will focus on preserving or enhancing the development speed while improving the ability of developers to rapidly find relevant information and communicate effectively.

Acknowledgments. We thank all participants of the study for their time and willingness to share insights with us and the research community. We are also grateful to Leif Singer and to the anonymous reviewers for their valuable comments on earlier versions if this paper.

APPENDIX: Technical and Technological Challenges

We now provide a brief overview of several technical and technological issues mentioned by the study participants. In contrast to the interaction and communication issues, which were largely consistent across the participant group, no set of technical and technological issues arose consistently within the group. Instead, it appears that such issues are much more specific to the activities that each developer performs. For completeness, we give a concise list of these issues below.

Missing code analysis solutions and development environment enhancements:

1. Static analysis for enhancing the contextual information and error messages provided to the developer.
2. Static analysis for visualizing "where the thing that is going to be injected comes from" in injection-based frameworks.
3. Debugging support for injection-based frameworks.
4. Debugging, reasoning about and tracking flows across languages, e.g., when Java, JavaScript and XML are involved.
5. Dependency analysis for Python as well as for bash scripts.
6. Support around XML tooling.

Languages and language extensions:

1. A language that provides a better memory management solution, as "current garbage collectors are still not good enough, especially for low-latency systems".
2. Explicit support for low-latency processing in the commonly used languages, such as C++, or, alternatively, having extensive set of libraries that implement such support.

Run-time solutions:

1. Efficient run-time monitoring for systems with high-availability and reliability requirements.
2. Modeling and predicting behaviors, as well as possible failures, of highly-available and reliable systems.
3. Approaches for recreating concurrency-related failures (besides repeated runs).
4. Solutions for introducing updates into a running system that cannot be stopped for an upgrade "even for a few seconds".

Tools:

1. Tools for performance measurement that adapt to different precision levels.
2. Tools for recording all user activities *together with their associated data*, so that interactions with the tool can be recreated in the lab in a reliable manner.
3. Reliable systems for simulating highly-distributed environments on a local machine.

Others:

1. A new data encoding for supporting large-scale low-latency data processing in an efficient manner.
2. Storage and computation models for large systems. Systems that scale up and scale out very well in load and data size.
3. Approaches for eliciting meaningful early feedback for large-scale systems with very long launching time and time-to-production.
4. Approaches for simplifying complex system and dividing them into the "right" set of components.
5. Approaches for supporting legacy software and outdated technologies.
6. Approaches for developing and validating software that runs on multiple target operating systems and environments.
7. Advances in security.
8. Solutions for context-awareness.

9. REFERENCES

[1] M. Barnett, C. Bird, J. Brunet, and S. K. Lahiri. Helping Developers Help Themselves: Automatic Decomposition of Code Review Changesets. In *Proc. of the 37th International Conference on Software Engineering (ICSE'15)*, pages 134–144, 2015.

[2] O. Baysal, R. Holmes, and M. W. Godfrey. No Issue Left behind: Reducing Information Overload in Issue Tracking. In *Proc. of the 22nd International Symposium on Foundations of Software Engineering (FSE'14)*, pages 666–677, 2014.

[3] K. Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 1999.

[4] K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, et al. Manifesto for Agile Software Development, 2001.

[5] A. Begel and B. Simon. Novice Software Developers, All over Again. In *Proc. of the International Computing Education Research Workshop (ICER'08)*, pages 3–14, 2008.

[6] B. W. Boehm. A Spiral Model of Software Development and Enhancement. *IEEE Computer*, 21(5):61–72, 1988.

[7] F. Calefato and F. Lanubile. SocialCDE: a Social Awareness Tool for Global Software Teams. In *Proc. of the 9th Joint Meeting on the Foundations of Software Engineering (ESEC/FSE'13)*, pages 587–590, 2013.

[8] Y. C. Cavalcanti, I. do Carmo Machado, P. A. da Mota Silveira Neto, E. S. de Almeida, and S. R. de Lemos Meira. Combining Rule-based and Information Retrieval Techniques to Assign Software Change Requests. In *Proc. of the 29th International Conference on Automated Software Engineering (ASE'14)*, pages 325–330, 2014.

[9] M. E. Conway. How Do Committees Invent? *Datamation*, 14(4):28–31, 1968.

[10] J. Corbin and A. Strauss. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*. Sage Publications, Inc., 3rd edition, 2008.

[11] D. Damian, L. Izquierdo, J. Singer, and I. Kwan. Awareness in the Wild: Why Communication Breakdowns Occur. In *Proc. of the 2nd International Conference on Global Software Engineering (ICGSE'07)*, pages 81–90, 2007.

[12] R. DeLine, M. Czerwinski, B. Meyers, G. Venolia, S. M. Drucker, and G. G. Robertson. Code Thumbnails: Using Spatial Memory to Navigate Source Code. In *Proc. of the 2006 Symposium on Visual Languages and Human-Centric Computing (VL/HCC'06)*, pages 11–18, 2006.

[13] K. C. Desouza, A. Chattaraj, and G. Kraft. Supply Chain Perspectives to Knowledge Management: Research Propositions. *J. Knowledge Management*, 7(3):129–138, 2003.

[14] P. Diebold and A. Vetro. Bridging the Gap: SE Technology Transfer into Practice: Study Design and Preliminary Results. In *Proc. of the International Symposium on Empirical Software Engineering and Measurement (ESEM'14)*, pages 52:1–52:4, 2014.

[15] B. Glaser and A. Strauss. *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Observations (Chicago, Ill.). Aldine de Gruyter, 1967.

[16] C. Godart, P. Molli, G. Oster, O. Perrin, H. Skaf-Molli, P. Ray, and F. A. Rabhi. The ToxicFarm Integrated Cooperation Framework for Virtual Teams. *Distributed and Parallel Databases*, 15(1):67–88, 2004.

[17] C. Gutwin, R. Penner, and K. A. Schneider. Group Awareness in Distributed Software Development. In *Proc. of the 2004 Conference on Computer Supported Cooperative Work (CSCW'04)*, pages 72–81, 2004.

[18] J. E. Hannay and H. C. Benestad. Perceived Productivity Threats in Large Agile Development Projects. In *Proc. of the International Symposium on Empirical Software Engineering and Measurement (ESEM'10)*, 2010.

[19] M. T. Hansen. The Search-transfer Problem: the Role of Weak Ties in Sharing Knowledge across Organization Subunits. *Administrative science quarterly*, 44(1):82–111, 1999.

[20] J. D. Herbsleb. Global Software Engineering: the Future of Socio-technical Coordination. In *Proc. of the International Conference on Software Engineering (ICSE'07), Track on the Future of Software Engineering (FOSE)*, pages 188–198, 2007.

[21] J. D. Herbsleb and R. E. Grinter. Architectures, Coordination, and Distance: Conway's Law and beyond. *IEEE Software*, 16(5):63–70, 1999.

[22] J. D. Herbsleb and R. E. Grinter. Splitting the Organization and Integrating the Code: Conway's Law Revisited. In *Proc. of the 21st International Conference on Software Engineering (ICSE'99)*, pages 85–95, 1999.

[23] M. Hewner and M. Guzdial. What Game Developers Look for in a New Graduate: Interviews and Surveys at One Game Company. In *Proc. of the Technical Symposium on Computer Science Education (SIGCSE'10)*, pages 275–279, 2010.

[24] S. Jalali and C. Wohlin. Agile Practices in Global Software Engineering - A Systematic Map. In *Proc. of the 5th International Conference on Global Software Engineering (ICGSE'10)*, pages 45–54, 2010.

[25] C. Jang, C. Steinfield, and B. Pfaff. Virtual Team Awareness and Groupware Support: an Evaluation of the TeamSCOPE System. *Int. J. Hum.-Comput. Stud.*, 56(1):109–126, 2002.

[26] A. Jedlitschka, M. Ciolkowski, C. Denger, B. G. Freimut, and A. Schlichting. Relevant Information Sources for Successful Technology Transfer: a Survey Using Inspections as an Example. In *Proc. of the International Symposium on Empirical Software Engineering and Measurement (ESEM'07)*, pages 31–40, 2007.

[27] D. Kawrykow and M. P. Robillard. Non-essential Changes in Version Histories. In *Proc. of the 33th International Conference on Software Engineering (ICSE'11)*, pages 351–360, 2011.

[28] I. Kwan, M. Cataldo, and D. Damian. Conway's Law Revisited: The Evidence for a Task-Based Perspective. *IEEE Software*, 29(1):90–93, 2012.

[29] M. Lavallée and P. N. Robillard. Why Good Developers Write Bad Code: an Observational Case Study of the Impacts of Organizational Factors on Software Quality. In *Proc. of the International Conference on Software Engineering (ICSE'15)*, volume 1, pages 677–687, 2015.

[30] L. Layman, L. Williams, D. Damian, and H. Bures. Essential Communication Practices for Extreme Programming in a Global Software Development Team. *Information & Software Technology*, 48(9):781–794, 2006.

[31] T. C. Lethbridge. What Knowledge Is Important to a Software Professional? *Computer*, (5):44–50, 2000.

[32] Y. Li, J. Rubin, and M. Chechik. Semantic Slicing of Software Version Histories. In *Proc. of the 30th International Conference on Automated Software Engineering (ASE'15)*, pages 686–696, 2015.

[33] G. Mark, D. Gudith, and U. Klocke. The Cost of Interrupted Work: More Speed and Stress. In *Proc. of the Conference on*

Human Factors in Computing Systems (CHI'08), pages 107–110, 2008.

[34] L. Martie, T. D. LaToza, and A. van der Hoek. CodeExchange: Supporting Reformulation of Internet-Scale Code Queries in Context (T). In *Proc. of the 30th International Conference on Automated Software Engineering (ASE'15)*, pages 24–35, 2015.

[35] A. Mockus and J. D. Herbsleb. Cataldo. In *Proc. of the 24th International Conference on Software Engineering (ICSE'02)*, pages 503–512, 2002.

[36] L. Moreno, G. Bavota, S. Haiduc, M. D. Penta, R. Oliveto, B. Russo, and A. Marcus. Query-based Configuration of Text Retrieval Solutions for Software Engineering Tasks. In *Proc. of the 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE'15)*, pages 567–578, 2015.

[37] K. Muslu, L. Swart, Y. Brun, and M. D. Ernst. Development History Granularity Transformations. In *Proc. of the 30th International Conference on Automated Software Engineering (ASE'15)*, pages 697–702, 2015.

[38] H. V. Nguyen, M. H. Nguyen, S. C. Dang, C. Kästner, and T. N. Nguyen. Detecting Semantic Merge Conflicts with Variability-aware Execution. In *Proc. of the 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE'15)*, pages 926–929, 2015.

[39] T. T. Nguyen, H. V. Pham, P. M. Vu, and T. T. Nguyen. Recommending API Usages for Mobile Apps with Hidden Markov Model. In *Proc. of the 30th International Conference on Automated Software Engineering (ASE'15)*, pages 795–800, 2015.

[40] J. Noll, S. Beecham, and I. Richardson. Global Software Development and Collaboration: Barriers and Solutions. *Inroads*, 1(3):66–78, 2010.

[41] R. Padhye, S. Mani, and V. S. Sinha. NeedFeed: Taming Change Notifications by Modeling Code Relevance. In *Proc. of the 29th International Conference on Automated Software Engineering (ASE'14)*, pages 665–676, 2014.

[42] G. Petrosyan, M. P. Robillard, and R. D. Mori. Discovering Information Explaining API Types Using Text Classification. In *Proc. of the 37th International Conference on Software Engineering (ICSE'15)*, pages 869–879, 2015.

[43] S. Rastkar, G. C. Murphy, and A. W. J. Bradley. Generating Natural Language Summaries for Crosscutting Source Code Concerns. In *Proc. of the 27th International Conference on Software Maintenance (ICSM'11)*, pages 103–112, 2011.

[44] M. P. Robillard and R. DeLine. A Field Study of API Learning Obstacles. *Empirical Software Engineering*, 16(6):703–732, 2011.

[45] M. P. Robillard and R. J. Walker. An Introduction to Recommendation Systems in Software Engineering. In *Recommendation Systems in Software Engineering*, pages 1–11. 2014.

[46] I. Rus and M. Lindvall. Guest Editors' Introduction: Knowledge Management in Software Engineering. *IEEE Software*, 19(3):26–38, 2002.

[47] A. Sarma, Z. Noroozi, and A. van der Hoek. Palantír: Raising Awareness among Configuration Management Workspaces. In *Proc. of the 25th International Conference on Software Engineering (ICSE'08)*, pages 444–454, 2003.

[48] K. Schwaber and M. Beedle. *Agile Software Development with Scrum*. Pearson, 2002.

[49] A. D. Sorbo, S. Panichella, C. A. Visaggio, M. D. Penta, G. Canfora, and H. C. Gall. Development Emails Content Analyzer: Intention Mining in Developer Discussions (T). In *Proc. of the 30th International Conference on Automated Software Engineering (ASE'15)*, pages 12–23, 2015.

[50] I. Steinmacher, A. P. Chaves, and M. A. Gerosa. Awareness Support in Global Software Development: Systematic Review Based on the 3C Collaboration Model. In *Proc. of the 16th International Conference on Collaboration and Technology (CRIWG'10)*, pages 185–201, 2010.

[51] M.-A. D. Storey, L. Singer, B. Cleary, F. M. F. Filho, and A. Zagalsky. The (R) Evolution of Social Media in Software Engineering. In *Proc. of the International Conference on Software Engineering (ICSE'14), Track on the Future of Software Engineering (FOSE)*, pages 100–116. ACM, 2014.

[52] M. Twain. *Life on the Mississipi*. 1883. Chapter 15.

[53] J. Whitehead. Collaboration in Software Engineering: a Roadmap. In *Proc. of the International Conference on Software Engineering (ICSE'07), Track on the Future of Software Engineering (FOSE)*, pages 214–225, 2007.

[54] Wikipedia. Ashley Madison Data Breach. https://en.wikipedia.org/wiki/Ashley_Madison_data_breach.

[55] Wikipedia. Office of Personnel Management Data Breach. https://en.wikipedia.org/wiki/Office_of_Personnel_Management_data_breach.

[56] Wikipedia. Sony Pictures Entertainment Hack. https://en.wikipedia.org/wiki/Sony_Pictures_Entertainment_hack.