From Program Comprehension to People Comprehension

Andrew Begel
Microsoft Research
Redmond, WA, USA
Email: andrew.begel@microsoft.com

***Abstract*—Large-scale software engineering requires many teams to collaborate together to create software products. The problems these teams suffer trying to coordinate their joint work can be addressed through tools inspired by social networking. Social networking tools help people to more easily discover and more efficiently maintain relationships than is feasible using one-to-one or face-to-face interactions. Applying these ideas to the software domain requires new kinds and combinations of software program and process analyses that overcome intrinsic limitations in the accuracy of the underlying data sources and the ambiguity inherent in human relationships.**

***Keywords*-software process, human aspects**

I. Introduction

Building large software products is a difficult endeavor for software companies, both from a software engineering perspective and from a human resources perspective. Many software teams, each consisting of tens to hundreds of engineers, must collaborate together to produce the software components that make up the products that they ship. Some teams, especially inexperienced ones, are resistant to structured processes, and collaborate with others in an ad hoc manner. This often leads to miscommunication, misunderstandings, non-transparent decision making, and mistrust, resulting in dysfunctional inter-team relationships [1], [2]. These kinds of coordination problems are known to be critical factors leading to software project failures [3].

A lack of information about other teams' work can interfere with each team's ability to assess and mitigate the risk of depending on others. Much of the information engineers seek concerns their shared work artifacts, such as features, interfaces, schedules, bugs, and documentation [4]. These artifacts are usually accessible, since they are often stored in server-based software repositories, however, these kinds of repositories were designed more for efficient storage than for enabling engineers to easily locate and extract data meaningful to their inter-team coordination tasks [5]. An entire body of research hosted at the Conference on Mining Software Repositories has explored how to discover and extract meaningful information from software repositories and help engineers with the difficult task of making sense of the data.

We conducted a survey in the summer of 2009 at Microsoft to learn how Microsoft engineers (i.e., developers, testers, program managers, and architects) rank their most difficult-to-answer information needs around inter-team coordination. The results revealed several needs that were so important to the respondents that if a tool were available to solve these needs, it would have a significant positive impact on their daily work [6]. The needs they chose are reflected in the following three real scenarios that we heard in speaking with software team engineers and managers at Microsoft.

1) A change in someone else's code has broken mine. Who wrote this code? Is there a bug in his code, or is it in mine? Why did he choose to write the code that way? Is there a specification for it? Who wrote the specification? What requirements was the developer considering when he worked on the code? Is there a test case that could catch the problem I have? Has anyone filed a bug report about the problem? Is it already fixed for a future release? If the developer and his team are unaware of the problem, what evidence can I show to convince them to make a change that will help me out?
2) My team needs to prioritize its bug list for the next release in six weeks. The team has 22 developers, 20 testers, and 15 program managers who have worked together on this project for the past three years. While much of what we have done is recorded in our software repositories, several important decisions and events were never written down. We depend on 3 other team's libraries to build our product – two of our program managers liaise with those teams to keep us up to date on their progress. However, other than a team on the same floor that sends their own program manager to attend our weekly status meetings, we do not know which other teams depend on our software. How do we take into account our own needs and goals, the expectations we have of our dependencies, the needs of our known customers, and the needs of our unknown customers, to choose the bugs that we should definitely fix?
3) Our team's testers each have expertise in a particular area. We would like them to be notified whenever there is a change to the software that they would be most suited to test. The problem is that our team has 300 testers, the software has 50 million lines of code, and

neither the testers nor the code have been classified by topic area. Ultimately, when we get this notification problem solved for our own team, we would like to extend the notifications to a sample of the 10,000 users of our product who participate regularly on our public discussion forum and have access to download and try out weekly builds of our software. Accuracy in for both sets of people is paramount because my team's testing resources and my users' attention spans are limited, and we do not want them to waste their time trying our software if there are no changes that they would find interesting.

Without software support, these scenarios often are addressed simply through direct communication between the people involved. It is hard, however, to scale this method to large deeply-interconnected software projects. We must augment the human element with analytic help people discover, learn, analyze, communicate, and collaborate with their dependencies, without requiring the team members to adopt extra processes or do extra work.

II. Better Working Through Social Networking

We believe a promising approach to addressing coordination needs is to augment existing program analyses with social networking. With the rise in popularity of Web 2.0-based socially-oriented tools, engineers in industry expect to collaborate with and learn about their colleagues using weblogs, microblogs, wikis, newsfeeds, online profiles, and social networking. Our new tool, Codebook, takes advantage of this trend to build a social network that connects software engineers via their shared work artifacts by mining the artifacts from software repositories (e.g. code, bugs, test cases, specifications, documentation, builds, and discussions on mailing lists or web sites) and analyzing them with a variety of software program, process, quality, and organizational network comprehension tools [6], [7]. Applications built on the Codebook platform enable engineers to find and directly act on information discovered about other teams and their work.

Many of the analyses required to solve the questions asked in these particular scenarios have been explored before in the literature. However, we see two open challenges to putting them together in the right way. First, we need to improve our understanding of the human aspects of software development in order to identify which problems actually need solutions, and which problems, though juicy from a research perspective, would unfortunately have little impact even if solved perfectly [8]. To understand these problems, it helps to have a concrete description of the social and software processes around creating and maintaining large software systems: fixing bugs, negotiating agreements with other teams, deciding to take dependencies on others, managing everyone's expectations around shared work, communicating and signaling between team members, and interacting with people who play different engineering roles, have more or less political power, and embrace different priorities.

The second challenge is that the data found in software repositories is not always accurate [9]. Aranda's study of bugs in Microsoft Windows found that the people listed on a bug report did not always play the role the bug report indicated, and people who were critical to the bug being fixed were sometimes not listed at all. Combining bug reports with emails still resulted in misleading explanations for how and why issues were handled. Many open questions remain — is there a way to combine multiple lines of evidence from several repositories in order to triangulate the data and synthesize a more accurate story to explain an event? Which repositories are more reliable than others? Is it possible to put error bounds on the elements of an explanatory story to know which parts to trust, and which parts to discount?

III. Conclusion

Helping software engineers to work better together is vitally important to helping resolve the software industry's perennial coordination crises. By leveraging our ability to analyze and comprehend the relationship between engineers and their work-related artifacts, we can build tools to help engineers make sense of the information stored in their repositories and enable them to more effectively collaborate with their peers.

References

[1] A. Begel, "Effecting change: Coordination in large-scale software development," in *Proceedings of CHASE*, May 2008.

[2] A. Begel and N. Nagappan, "Coordination in large-scale software development: Helpful and unhelpful behaviors," Microsoft Research, Tech. Rep. MSR-TR-2009-135, September 2009.

[3] J. D. Herbsleb, A. Mockus, and J. A. Roberts, "Collaboration in software engineering projects: A theory of coordination," in *Proceedings of ICIS*, Milkwaukee, WI, 2006.

[4] A. Begel, N. Nagappan, C. Poile, and L. Layman, "Coordination in large-scale software teams," in *Proceedings of CHASE*, 2009, pp. 1–7.

[5] R. Holmes and A. Begel, "Deep intellisense: a tool for rehydrating evaporated information," in *Proceedings of MSR*, Leipzig, Germany, 2008, pp. 23–26.

[6] A. Begel, Y.-P. Khoo, and T. Zimmermann, "Codebook: Discovering and exploiting relationships in software repositories," in *Proceedings of ICSE*, Cape Town, South Africa, May 2010.

[7] A. Begel and R. DeLine, "Codebook: Social networking over code," in *Proceedings of ICSE, NIER Track*, Vancouver, BC, Canada, May 2009.

[8] L.-T. Cheng, C. de Souza, Y. Dittrich, M. John, O. Hazzan, F. Maurer, H. Sharp, J. Singer, S. E. Sim, J. Sillito, M.-A. Storey, B. Tessem, and G. Venolia, *Proceedings of the 2008 international workshop on Cooperative and human aspects of software engineering*. Leipzig, Germany: ACM, 2008.

[9] J. Aranda and G. Venolia, "The secret life of bugs: Going past the errors and omissions in software repositories," in *Proceedings of ICSE*, Vancouver, BC, Canada, 2009.