Collaborative Software Development in Ten Years: Diversity, Tools, and Remix Culture

– Position Paper –

Thomas Zimmermann
Microsoft Research
Redmond, WA, USA
tzimmer@microsoft.com

Christian Bird
Microsoft Research
Redmond, WA, USA
cbird@microsoft.com

ABSTRACT
Over the next ten years, collaboration in software engineering will change in a number of ways and research will need to shift its focus to enable and enhance such collaboration. Specifically, we claim that software in the small will become more popular and even large software will be built by fewer people due to better tools. For large projects, research will need to address the collaboration needs of project members other than just developers, including quality assurance engineers, build engineers, architects, and operations managers. Finally, code reuse and sharing will change as a result of a growing software remix culture, leading to more loosely coupled and indirect collaboration.

Author Keywords
Diversity, Tools, Collaboration, Remix Culture

ACM Classification Keywords
D.2.9 Software Engineering – Management, K.4.3 - Organizational Impacts – Computer-Supported Cooperative Work

INTRODUCTION
The past ten years has seen both an increase in the size of development teams and a proliferation of software research in the form of empirical studies providing insight and tools providing aid for the commensurate collaboration that is required in such projects. Despite these successes, there is still much to be done, both because there still exist chasms between what is needed and what has been provided *today* and because the software development landscape is changing rapidly, particularly in the mobile and web development spaces.

We claim that due to the collaborative tools that are available and recent shifts and fractures in software development, such as the focus on mobile applications and web development, the coordination needs in many projects will decrease. Despite this, there is a shrinking but still non-trivial amount of very large projects that require new forms of collaborative aid for differing roles in enterprise level software development efforts.

In this paper, we make the following assertions:

- Due to better tools and a growing market for smaller, specialized software, the size of software teams and in turn the coordination needs will *decrease* over the next ten years.
- For the projects that do continue to grow in terms of team size, diversity and specialization will increases, leading to a need for collaboration between project members other than just developers.
- Due to a growing remix culture in software development, some forms of collaboration will be indirect and more loosely coupled due to sharing between multiple parties without necessarily having direct interaction or even awareness.

FEWER PEOPLE WILL BUILD LARGER SOFTWARE – THANKS TO BETTER TOOLS
Over the past decades, the tools, programming languages, and APIs to build software have dramatically improved. IDEs such as Microsoft Visual Studio or Eclipse come with wizards and designers that help with many routine tasks, for example to create template projects of a certain type or to simplify the interaction with a database. Many languages now have syntactic sugar (such as extension methods and lambda expressions) and availability of APIs has increased. Service-oriented architectures now facilitate the reuse of functionality across applications and allow the mash up of new software by combining existing services. Tools for program understanding, debugging, collaboration, and coordination have substantially improved and will continue to improve over the next decade. Search engines and collaborative question and answer sites like Stack Overflow have changed the way that developers seek answers to any programming problems.

At the same time the complexity and size of software has increased substantially. For example, the size of Microsoft Windows grew from 5M to 50M lines of code between 1993 and 2003 [1]. For open source, Deshpande and Riehle found that the "total amount of source code and the total number of projects double about every 14 months" [2].

We believe that this growth of software will not continue forever. Further, small and medium sized projects will become more common (e.g., mobile apps). However, at the same time, we expect that tool and technology support will

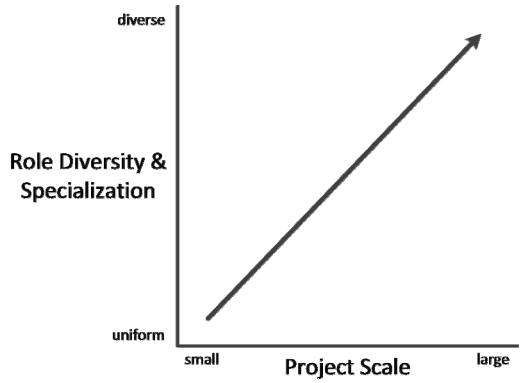

Figure 1. The relationship between project size and the diversity and specialization that occurs.

continue to improve in the next few years. *Thanks to better tools, fewer people are needed to build large(r) software systems.* This will have several implications on collaborative software development:

- As fewer people will be needed to build software, the coordination needs will become less critical. Often simple email will be sufficient to coordinate smaller teams.
- We also expect that for some teams coordination across different roles (developer, build engineer, operations) will become more important. Further, organizational structures will become more fluid as they will be easier to change thanks to better tool support.
- As the tools for program understanding and information seeking will improve, people will have to rely less on other people. Rather than finding an expert and waiting for a response, they can query a tool and get immediate answers. This will improve the speed of software development.

DIVERSITY AND SPECIALIZATION OF ROLES

The vast majority of software engineering research today focuses on the role of developers and software implementation, maintenance, and testing. While these are critical parts of any software development project, we have observed that other roles and activities which are vital to the success of large development projects have been largely neglected.

As we pointed out in the last section, the size of software development projects in the future will be both larger and smaller than they are today. As shown in Figure 1, for those projects that are large in scope (e.g. Microsoft Windows or Adobe Photoshop), there is a necessary diversity and specialization that occurs in project roles and individual developers. These roles do not involve writing code, but still require collaboration and coordination.

To illustrate such roles in software development projects that require focus from research, we briefly survey important roles in software development that receive little attention, but require collaboration.

Quality assurance (QA) engineers are primarily concerned with tasks that assess software quality and dealing with issues that these tasks uncover. In some cases, such as failed unit tests, triaging the problem, identifying the person responsible for fixing the problem, and communicating critical details are all straightforward tasks. We have observed at Microsoft that when these tasks are not straightforward, there is little tool support to aid these engineers and many hours can be spent doing detective work to deal with just one issue. A common example is performance testing. If a product passed the performance tests yesterday, but not today, the assurance engineer must ascertain what caused the slowdown, communicate with the team responsible for the change (if in fact, it was the result of one change and not an intermingling of multiple changes), and help them determine the actions to take in order to solve the problem based on the facts available to the QA engineer.

Build engineers have the task of continuously building the product and running tests on the resulting builds. The build infrastructure for a large project can be nearly as complex as the product itself and can impact product development [3]. Developers rely on builds in their daily work. Testers rely on regular builds so that functionality can be tested. Clearly collaboration is required between build engineers and other roles as a project continuously changes.

Most non-trivial projects use branches (also known as workspaces) to divide up development, with code "moving" between branches when a set of functionality is stable and complete. Moving code too early risks adversly affecting teams due to poor quality and breaks. Waiting too long can slow down teams due to dependencies in the development cycle. What should the branching structure in a project look like and how should code move through the system between teams? The role of people making these decisions and moving code, whether formal or informal, can have a dramatic effect on the pace of the project. Research will need to focus on these questions in order to aid these project members.

Program/project managers [4] facilitate coordination between stakeholders. They are responsible for drafting specifications for developers and testers, monitoring progress of different parts of the software system, and relaying that information to upper management. While there has been some effort towards providing analytic tools for these individuals, the state of research today is that we often don't even know what should be measured and reported to help them to do their jobs effectively.

For space reasons, we do not elaborate on a number of other important roles such as those within operations, which become more important with the growth of software as a service, software architects who plan the high level of the system but don't produce code, release managers, and technical writers. These serve to illustrate that not only are

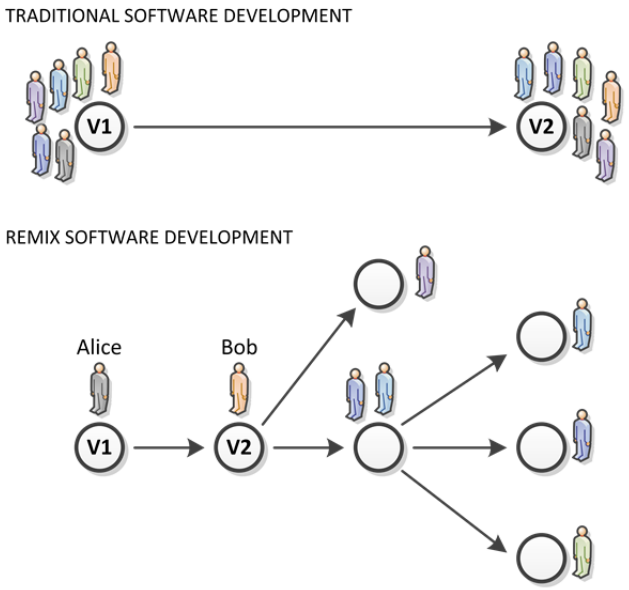

Figure 2. Illustration of the difference between traditional and remix software development. In the latter, fewer people (who may or may not know about each other) work on remixed versions, but there will be more versions and variants.

there many roles whose collaboration needs have not been examined, but also that the *types* of collaboration that occur in large software projects is quite diverse.

REMIX DEVELOPMENT

We also expect that the style of software development will change over the next years. Today, software is developed by teams who collaboratively work on one version of the software and implement new features. However, with the increased availability of open source, sharing and building on existing code will become more common. We expect that the cost of forking code will be reduced thanks to distributed version control systems such as Git and Mercurial, which allow selective integration of changes. We believe that this will lead to more frequent, smaller releases of software by different groups of people, often even individuals. Because of the similarity to remix culture [5], which allows and actually encourages derivative works, we term this new development style *remix software development*.

Figure 2 further illustrates the difference to traditional software development. In remix development, people will share code with others who then use this code to derive new versions of the same or another project. We expect that fewer people will work on a remixed version but ultimately there will be more versions and variants (forks) of a project, often even for individual features. While in traditional development team members typically know each other (either personally or electronically), in remix development two people (e.g., Alice and Bob) do not necessarily need to know about each other but technically collaborate on the same software.

There has been already first evidence of a remix culture in software development. For example many developers now use code examples from Stack Overflow,[1] a question-and-answer website on computer programming, in their software. Another example is scripts, which are often shared and customized across people. The TouchDevelop[2] app for Windows phones actively facilitates the remix of scripts with a Script Bazaar that allows users to share, reuse, and publish the derived scripts.

In remix software development, people will collaborate in the sense that they use and build on each other's code, but actually have no or little need of coordinating. Future collaboration tools should include support for remix activities.

BIOGRAPHIES

Christian Bird is a researcher in the Research in Software Engineering Group at Microsoft Research. He is primarily interested in how groups work together to develop software and his area of research is empirical studies of collaborative activities in software projects. He has studied software development in the open source realm, at IBM, and at Microsoft.

Thomas Zimmermann is a researcher in the Research in Software Engineering Group at Microsoft Research. His research interests include empirical software engineering, mining software repositories, software reliability, development tools, recommender systems, and social networking. He is best known for his work on systematic mining of version archives and bug databases to conduct empirical studies and to build tools to support developers and managers.

To learn more about the research of both authors, please visit: http://research.microsoft.com/en-us/groups/ese/

ACKNOWLEDGEMENTS

Thanks to the reviewers for their insightful comments.

BIBLIOGRAPHY

[1] O'Brien, L. How Many Lines of Code in Windows? *Knowing.Net* (December 2005).

[2] Deshpande, A. and Riehle, D. The total growth of open source. *Open Source Development, Communities and Quality* (2008), 197--209.

[3] McIntosh, S., Adams, B., Nguyen, T.H.D., Kamei, Y., and Hassan, A.E. An empirical study of build maintenance effort. In *33rd International Conference on Software Engineering* (2011), 141--150.

[4] Sinofsky, S. *PM at Microsoft*. http://blogs.msdn.com/b/techtalk/archive/2005/12/16/504872.aspx, 2005.

[5] Lessig, L. *Remix: Making Art and Commerce Thrive in the Hybrid Economy*. The Penguin Press HC, 2008.

[1] http://stackoverflow.com/
[2] http://www.touchdevelop.com/