Sociotechnical Coordination and Collaboration in Open Source Software

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

***Abstract*—Over the past decade, a new style of software development, termed open source software (OSS) has emerged and has originated large, mature, stable, and widely used software projects. As software continues to grow in size and complexity, so do development teams. Consequently, coordination and communication within these teams play larger roles in productivity and software quality. My dissertation focuses on the relationships between developers in large open source projects and how software affects and is affected by these relationships. Fortunately, source code repository histories, mailing list archives, and bug databases from OSS projects contain latent data from which we can reconstruct a rich view of a project over time and analyze these sociotechnical relationships. We present methods of obtaining and analyzing this data as well as the results of empirical studies whose goal is to answer questions that can help stakeholders understand and make decisions about their own teams. We answer questions such as "Do large OSS project really have a disorganized bazaar-like structure?" "What is the relationship between social and development behavior in OSS?" "How does one progress from a project newcomer to a full-fledged, core developer?" and others in an attempt to understand how large, successful OSS projects work and also to contrast them with projects in commercial settings.**

I. Introduction

Open source is a popular and growing method of development that has produced software that rivals and in some cases even exceeds the scale and quality of traditional software projects. The last few years has even seen an embrace of open source projects and methdologies by the commercial world. But how do the large and successful open source projects produce high quality artifacts outside the policies, mandates, and management that accompany more traditional, industrial development contexts? Development in the large is a collaborative enterprise and if the success and impact of work such as Peopleware [1] is any indication, the *people* and the ways in which they work together play a large factor in the success or demise of any sizable software project. Like Conway [2] I believe that there is also a strong relationship between the technical and social interactions that occur in any software project. Understanding these are of importance in determining how these projects work and avoiding failure in the future.

My dissertation was primarily focused on studying large (in terms of people and code) open source projects and the interplay betweeen the processes that goes on in them. In this paper, I summarize the techniques, analysis, and questions that were introduced and at least partially answered during my graduate work at U.C. Davis. I stress that while this work comprised my dissertation, like most successful research, it was performed in concert with many others who I am in debt to and acknowledge in this paper (and note that I use the pronoun "We" rather than "I" not by convention, but out of recognition of those who this work could not have been done without).

We use communication and coordination data from mailing lists, source code repository histories, and bug tracking databases to characterize the relationship between participants social and development behavior and examine the social structure that exists in large OSS projects. For instance, we have found that OSS projects do not typically follow a "bazaar"-like organization and instead tend to organically organize into teams. We examined the process by which OSS project newcomers enter a community and identified the factors of importance for an OSS participant migrating from a bystander to a full fledged, core developer.

Quality (usually measured in defects of one kind or another) is one of the most important aspects of any software project and we therefore investigated the relationship between quality and project coordination & organization. We showed that collaboration history combined with technical relationships (such as dependencies within software components) can be used to predict which components of a system will be the most failure prone with higher accuracy than previous methods. We studied distributed development in both open source and commercial projects, characterized the level of geographic and organizational distribution, and examine the relationship between distributed development and software quality in a number of contexts, leading to a higher level, aggregate theory of distribution and quality. And finally, We performed an analysis of code ownership on the same three projects and show how the relationship between ownership and quality is affected by the development style used.

Taken together, these data-gathering methods and the empirical studies that they enabled provide an understanding of the collaboration processes at work in OSS (and in some cases commercial) projects and also give researchers tools and techniques that they can use to gather and analyze data to answer their own related questions.

One possible misconception in studying open source software is that it encompasses just *one* method or process for developing software. In reality, the processes used and development methodology in open source is as (or more) varied as software projects in commercial contexts. However, there are categories of project types and Berkus [3] discusses five types of project

organizations in OSS. In my dissertation, we made an effort to cover different types of projects. For example Apache is a *foundation* with a well-organized, hierarchical governance structure and formalized policies. Postgres is a *community*, more informal, with consensual group decision making and Python is *monarchist*. Please see the original work for threats to validity and contexts in which we believe the results may or may not hold.

II. Mining Open Source Data

To study and perform emprirical analysis on OSS projects, one first has to obtain data from these projects relative to the questions that one hopes to answer.

Previous research exists that presents methods of gathering historical development behavior from source code repositories [4] as well as issue tracking systems [5] and even linking the data between them [6]. However, the dominant form of coordination within OSS is *email communication* [7], [8]. In my dissertation, we developed techniques for gathering collaborative behavior from mailing list archives and analyzing this behavior in a number of ways.

A. Mining Email Social Networks

Most OSS projects archive their mailinglist interactions because they provide a valuable resource to newcomer project members as well as a historical reference for what decisions were made and why. Fortunately, obtaining these archives and parsing them is not difficult, and we were able to obtain these for a number of prominent projects including Apache, Perl, Python, Postgres, and Ant.

Mailinglist messages contain vital information in their headers, including *who* sent the message, *when* it was sent, and *what* message it is in response to. In addition, the content of the message is the actual payload and is amenable to easy lightweight analysis as well.

One non-trivial problem that crops up when analyzing mailinglists is the issue of *email aliasing*. Many developers use multiple email addresses (especially over a period of many years) and for accurate analysis, we need to be able to attribute all messages sent by a participant to that one participant and not multiple personas. For example the developer *Ian Holsman* uses 7 different email aliases, including `ian.holsman@cnet.com`, `ianh@holsman.net`, and `ianh@apache.org`. Sometimes aliases have very little relationships to developers: the developer Ken Coar uses the name *Rodent of unusual size* associated with email address `ken.coar@golux.com`.

We developed a clustering method based on email similarity and email naming conventions to identify aliasing candidates [9]. The results of this clustering still require manual post-processing, but greatly reduces the amount of work required.

Once aliases have been removed, we determine who was talking about what and who has responded to who. These threads of messages between developers create social networks of their communication activity. From this point, techniques such as Social Network Analysis [10] provided valuable information about individual developers' roles in a project community as well as the community as a whole that was used in subsequent studies.

B. Mining Work Contributions

The *content* of the messages are just as important as the message metadata. Project participants who do not have write access to the source code can only contribute code in the form of *patches* and even core developers often post patches for review. Submitting a patch to a project mailinglist is evidence of project expertise, an investment in time, and a willingness to contribute. We developed methods of mining these patch contributions from mailing lists *and* determining if they were accepted to the project, even in the presence of edits to the patches prior to applying them to the source code repository [11]. This data was valuable in a subsequent study on OSS project immigration phenomena [12].

III. Sociotechnical Dynamics

Having developed mining techniques and mined communication data along with software repository data, we were able to conduct a number of empirical studies to answer questions that we and others have posed regarding how OSS projects actually work. We highlight our key results here.

A. How are social interaction and development behavior related?

As an activity that involves hundreds and in some cases, thousands of people, we believe that OSS project development is an inherently social process. To investigate this belief, we gathered both social (mailing list) and technical (source code repository) historical data. We used social network analysis to identify the key participants in the communication social network in apache using *betweenness centrality*, a global measure of network topological importance and *degree centrality*, a more local measure [10], [9], and quantitively examined the relationship between these measures and development behavior. Figure 1 shows a social network from Apache that is derived only of participants that send at least 150 messages. In a network this small, it is easy to identify the important participants, but the Apache mailing list has hundreds of participants.

We used standard statistical analysis, and details can be found in the original paper [9]. In short, we did find a strong relationship between development behavior and the level of importance that participants have in the social network.

In this study we found that:

- Participants who are core developers (have write access to the project repository) have positions of higher importance in the project social network than others.
- Both measures of network topological importance show a strong positive relationship with development activity. Developers who act as information brokers tend to be those that contribute the most.
- *Betweenness centrality*, a measure of *global* importance, is a better indicator of development behavior than *degree centrality*.
- There is a much lower correlation between documentation changes and social network importance than between source code changes and social network importance. Documentation contributors do not coordinate with others at a high level.

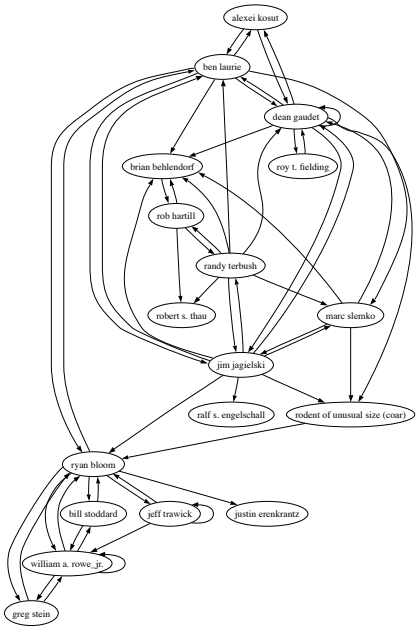

Figure 1: Pruned Social Network of Apache Emailers (Each link indicates at least 150 messages sent, or replied-to).

B. How does a project newcomer become a core developer?

Each OSS project has a core team of developers who have the authority to commit changes to the repository; this team is the elite core of the project, selected through a meritocratic process from a larger number of people who participate on the mailing list. Understanding the factors that influence the "who, how and when" of this process is critical for the sustainability of OSS projects and for outside stakeholders who want to gain entry and succeed. Prior research indicates that certain types of behaviors, such as the duration and intensity of participation on the developer mailing list, and the submission of patches, play a role in immigration [13], [14], [15]. However, this work has largely been qualitative and/or descriptive. We used quantitative hazard rate modeling, which supports the statistical testing of hypotheses concerning the influence of these factors on the rate of immigration [12]. We developed a theory of open source project joining, and used data from Postgres, Python, and the Apache web server to statistically evaluate this theory using a piecewise-constant proportional hazard rate model. Quantitative modeling reveals variations across the projects in the effects of a participant's a) duration in an OSS community; b) their volume of patch submission; and c) and their social status. These variations can be attributed to differences across the projects in institutional norms for joining, technical complexity of the projects and governance mechanisms.

In both Apache and Postgres, the statistical models supported the hypothesis that the rate of promotion to core developer first increases, and then decreases with tenure time (see Figure 2). In all three cases, the data, when plotted, shows this trend; however, in Python the results are not statistically significant. The difference may be due to the centralized community structure and more ad hoc immigration policies in this *monarchist* project or could be attributable to the calendar duration of the projects: Python is 4 years younger than both

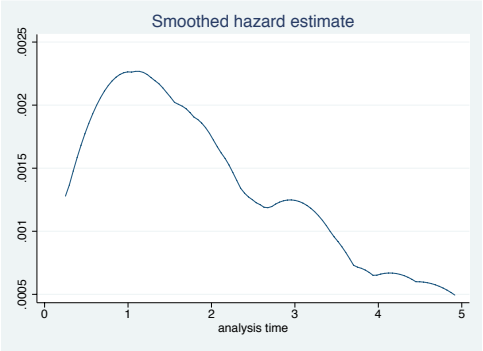

Figure 2: Fitted hazard rate for immigration events (promotions to core developer) in postgres by project membership time in years. Note an initial peak around 1 year followed by a drop-off.

Apache and Postgres; perhaps the community's reaction to newcomers is still evolving.

In Python and Postgres, prior history of patch submission has a significant effect positive effect. The effect is positive and within the same order of magnitude, but not statistically significant in Apache. We thus conclude that demonstrated skill level via patch submission plays an important role in Python and Postgres, but results are inconclusive in Apache. The effect in Python is especially strong. This is consistent with stated institutional norms of the Python project, which emphasize display of skills through patch submissions and other technical contributions as a way of gaining status.

The social network measure, indegree, which is a measure of the breadth of response to an individual, and thus status within the community also had a significant effect on immigration, although the effect is moderate. This is especially interesting given the varied governance structures and levels of formality with regard to the immigration process in the projects. The effect of social network status is specially strong in Postgres, reflecting Josh Berkus's description [3] of Postgres as a *community* project, where decisions are made communally. Still, the significance in all projects indicates a phenomenon that may generalize well to a significant portion of other OSS projects.

C. Are OSS communities haphazard and "bazaar" like or are they more organized?

Commercial software project managers design project organizational structure carefully, mindful of available skills, division of labour, geographical boundaries, etc. We contrasted these organizational "cathedrals" with the "bazaar-like" nature of Open Source Software (OSS) Projects, which have no pre-designed organizational structure (referencing Raymond's famous essay [7]. Any structure that exists is dynamic, self-organizing, latent, and usually not explicitly stated. Still, in large, complex, successful, OSS projects, we do expect that subcommunities will form spontaneously within the developer teams. Studying these subcommunities, and their behavior can shed light on how successful OSS projects self-organize. This phenomenon could well hold important lessons for how commercial software teams might be organized. Building on known well-established techniques for detecting *community*

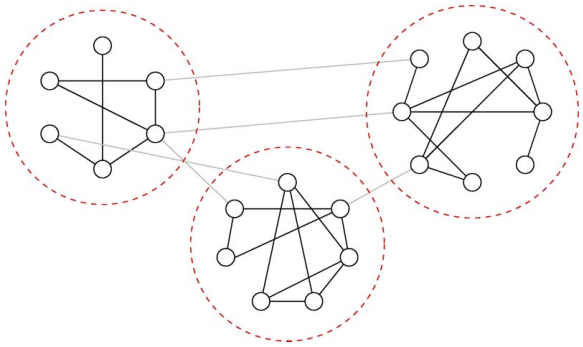

Figure 3: An example of community structure within an email social network.

structure in complex networks [16], we extracted and studied latent subcommunities from the email social network of several projects: Apache HTTPD, Python, PostgresSQL, Perl, and Apache ANT [17]. We then validated them with software development activity history.

We found clear subcommunities (teams) in all of the projects studied. These became even more clearly delineated when constraining the communication that we used in our analysis to messages directly mentioning product topics, *viz.,* emails that specifically name actual code artifacts. These communications require more technical expertise and are separate from messages that discuss more broadly accessible topics such as release schedules and project-wide policies (we term these "process" messages).

In four of the five projects, developers worked together on the same file with people in their own subcommunity much more often than people in others on average. This indicates that the communication behavior is tied to their collaborative development activities.

We also examined the development and communication activities of people in the groups identified to see if they were in fact working together on common tasks. Due to the sheer number of groups identified over the life of all five projects, a comprehensive manual inspection was not possible. However, we randomly sampled and found that the groups' activities reflected focused efforts towards specific goals. We refer the reader to our original study for details on these case studies [12].

IV. Relationship with Quality

A. Can social and technical relationships help identify failure prone components?

Studies have shown that social factors in development organizations have a dramatic effect on software quality [18], [19], [20]. Separately, program dependency information has also been used successfully to predict which software components are more fault prone. Interestingly, the influence of these two phenomena have only been studied in isolation.

Intuition and practical experience suggests, however, that task assignment (i.e. who worked on which components and how much) and dependency structure (which components have dependencies on others) together interact to influence the quality of the resulting software. We argue that these forms of information should be used together. The intuition behind our approach is that software components may be related through important but *different* types of relationships. By aggregating these relationships our ability to predict failures will increase. We do this in two ways.

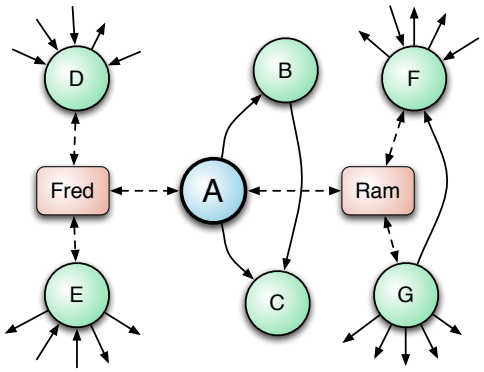

Figure 4: An example sociotechnical network. Circles are components and solid lines, dependency relationships. Rectangles are developers and dashed lines represent contributions made.

We studied the influence of combined *socio-technical software networks* on the fault-proneness of individual software components within a system [21]. An example of an socio-technical network is shown in Figure 4. The network properties of a software component in this combined network were able to predict if an entity is failure prone with greater accuracy than prior methods which use dependency or contribution information in isolation. We evaluated our approach in different settings by logistic regression defect prediction models on Windows Vista and across six releases of the Eclipse development environment including using models built from one release to predict failure prone components in the next release. We compared this to previous work. Results of our empirical study showed a strong correlation between the centrality of software components and the number of post-release failures. In every case, our method performed as well or better and was able to more accurately identify those software components that have more post-release

B. What is the affect of distributed development on software quality?

Existing literature on distributed development in software engineering, and other fields discuss various challenges, including cultural barriers, expertise transfer difficulties, and communication and coordination overhead [22], [23], [24], [25], [26]. Conventional wisdom, in fact, holds that distributed software development is riskier and more challenging than collocated development. While there are studies that have examined the delay associated with distributed development and the direct causes for them [27], there has been much less attention (See *e.g.,* [28]) to the effect of distributed development on software quality in terms of post-release failures. We evaluate this belief, empirically studying the overall development of Windows Vista [29] as well as Firefox and Eclipse [30] comparing the post-release failures of components that were developed in a distributed fashion with those that were developed by collocated teams.

In Vista, we found a negligible difference in failures [29]. This difference becomes even less significant when controlling for the number of developers working on a binary. Furthermore, we also found that component characteristics (such as code churn, complexity, dependency information, and test code coverage) differ very little between distributed and collocated components. Based on discussions with stakeholders, and the software process used during the Vista development cycle, we enumerated certain practices in place that may have mitigated some of the difficulties of distributed development.

We identified the top contributors that made 95% of the changes over multiple major releases of FIREFOX and ECLIPSE and determined their geographic locations and organizational affiliations [30] We found that FIREFOX is both organizationally and geographically distributed with over a third of its components receiving major contributions from developers on different continents. Although over half of contributions come from the California bay area (San Francisco and surrounding region), these come from a myriad of commercial organizations such as Google, Intel, and Red Hat. Interestingly, components that are highly distributed have no more defects than those that are not. In contrast, We found that ECLIPSE does not fit the typical open source project profile. ECLIPSE is directed and developed largely by one company; with IBM making 96% of the total commits (49% coming from one lab in Ottawa, Canada). It is also not largely distributed as 85% of the plugins can trace $\frac{3}{4}$ of the commits to one development site. Further, software components in ECLIPSE that are geographically distributed have far more post-release bugs than those whose changes originate primarily at one site.

Although we have only studied three projects in depth (Vista, FIREFOX, and ECLIPSE), based on our findings and also discussions with managers and developers involved in geographically distributed development, we have developed a theory regarding quality:

Software projects which have many distributed components will put measures in place to deal with the associated difficulties of distribution, thus mitigating the effect of such barriers on quality. In contrast, projects which are largely collocated lack such processes and policies, and the few distributed components will suffer greatly in terms of quality

C. Does ownership and expertise affect software quality?

Ownership is a key aspect of large-scale software development. We examined the relationship between different ownership measures and software faults/failures in three large software projects drawn from different process domains: Windows Vista, Windows 7, the ECLIPSE Java IDE, and the FIREFOX Web Browser. We found that in all cases, different measures of ownership such as the number of low-expertise developers, and the proportion of ownership for the top owner have a relationship with both pre-release faults and post-release failures [31]. However, we find that the strength of the effects is related to the development process used. Vista shows the strongest relationship with ownership level, followed by ECLIPSE, and then Firefox, suggesting that the more that a project uses an open source style process, the more that team sizes rather than ownership levels affect to failures.

We evaluated three measures of ownership by examining their effects when controlling for code metrics known to have a relationship with failures: size, complexity, and churn. We use ownership as a proxy for expertise as followed by others [32], [33], [34] and evaluate the hypothesis that more changes by those with low expertise leads to more failures [35].

For each component we count the number of contributions and divide the proportion of total contributions down by contributing developer. Thus if `foo.dll` had 100 changes made and Clara made 40 of those changes, Clara's ownership of `foo.dll` is 40%

Ownership: The ownership of the top contributing developer for a particular software component is considered the ownership of the component. Higher ownership means that more commits were made by a developer with expertise.

Minor Contributor: A developer who has made changes to a component, but who made less than 5% of the commits to a particular component has low expertise. [1].

Major Contributor: A developer who has made changes to a component and whose ownership is at or above 5% is a major contributor to the component and has a non-trivial amount of experience with the component.

After accounting for size, complexity, and code churn, there was a clear trend of ownership having a stronger relationship to failures in Vista and 7 than in ECLIPSE and stronger in ECLIPSE than FIREFOX. More minor contributors means more failures and a higher level of component ownership leads to fewer failures in all cases. In addition, across all projects, the effect of major contributors on quality was weak and often not statistically significant, indicating that the number of higher-expertise contributors has little effect on quality. In the context of Windows, where formal ownership policies are in place, the violation or adherence to such policies had a strong effect on software quality. In the two projects without such policies, we see an effect, but it is clearly not as strong.

In a deeper investigation into Windows, the project where the effect was the strongest, we found that 52% of the components had minor contributors who were major contributors to other components that the original had a dependency with. Thus, one common reason that a developer is a minor contributor to a component is that he is a major contributor to a depending component.

The major benefit of these findings is that this is an actionable result. For organizations where ownership has a strong relationship with defects (which should be easy to identify by replicating our lightweight analysis), we present the following recommendations. These are currently being evaluated at Microsoft.

1. Changes made by minor contributors should be reviewed with more scrutiny.

2. Potential minor contributors should communicate desired changes to developers experienced with the respective binary.

[1] A sensitivity analysis with threshold values ranging from 2% to 10% yielded similar results.

3. Components with low ownership should be given priority by QA resources.

V. Conclusion

These results have begun to shed light on how large, successful open source projects are able to coordinate their work. In some cases, we have been able to compare projects from open source and commercial domains. However, like all research, these findings raise additional questions.

How well do these results generalize? What tools or processes can enhance coordination and communication in these projects? As the lines between open source and commercial projects blur (e.g. consider how much open source code ships on Cisco routers these days), what additional coordination mechanisms will be needed and how will OSS projects, or the consumers of OSS projects cope? Clearly, as the software process landscape changes, so do the pertinent questions that need to be addressed.

Acknowledgment

I would like to thank and acknowledge my advisor, Prem Devanbu, for his constant insight, advice, support, motivation, patience, and pretty much everything else during my graduate career.

I would also like to acknowledge those who I was lucky enough to work with during my Ph.d.

- Harald Gall, Abraham Bernstein, and Adrian Bachmann at the University of Zurich
- Nachiappan Nagappan, Brendan Murphy, Tom Zimmermann, and Andy Begel during two internships at Microsoftt Research
- Clay Williams, Patrick Wagstrom, Peri Tarr, and Tim Klinger during an internship at IBM Research
- Peter Rigby and Daniel German at the University of Victoria
- Prem Devanbu, Zhendong Su, Vladimir Filkov, Raissa D'Souza, Anand Swaminathan, Greta Hsu, Earl Barr, Daryl Posnett, Eirik Aune, Patrick Duffy, Alex Gourley, Zach Saul, David Pattison, Foyzur Rahman, and Roozbeh Nia at the University of California, Davis

References

[1] T. DeMarco and T. Lister, *Peopleware: productive projects and teams*. Dorset House Publishing Co., Inc. New York, NY, USA, 1987.

[2] M. Conway, "How do committees invent," *Datamation*, vol. 14, no. 4, pp. 28–31, 1968.

[3] J. Berkus, "The 5 types of open source projects," 2007, march 20, 2007 http://www.powerpostgresql.com/5_types.

[4] T. Zimmermann, A. Zeller, P. Weissgerber, and S. Diehl, "Mining version histories to guide software changes," *Software Engineering, IEEE Transactions on*, vol. 31, no. 6, pp. 429–445, 2005.

[5] M. Fischer, M. Pinzger, and H. Gall, "Populating a release history database from version control and bug tracking systems," in *ICSM '03: Proceedings of the International Conference on Software Maintenance*. Washington, DC, USA: IEEE Computer Society, 2003, p. 23.

[6] J. Śliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?" in *The workshop on Mining Software Repositories*, 2005.

[7] E. S. Raymond, *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. Sebastopol, California: O'Reilly and Associates, 1999.

[8] K. Kuwabara, "Linux: A bazaar at the edge of chaos," *First Monday*, vol. 5, no. 3, March 2000. [Online]. Available: http://www.firstmonday.org/issues/issue5_3/kuwabara/index.html

[9] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan, "Mining email social networks," in *Proceedings of the 3rd International Workshop on Mining Software Repositories*, 2006.

[10] S. Wasserman and K. Faust, *Social network analysis: Methods and applications*. Cambridge University Press, 1994.

[11] C. Bird, A. Gourley, and P. Devanbu, "Detecting Patch Submission and Acceptance in OSS Projects," in *Proc. of the 4th International Workshop on Mining Software Repositories*, 2007.

[12] C. Bird, A. Gourley, P. Devanbu, A. Swaminathan, and G. Hsu, "Open borders? immigration in open source projects," in *MSR '07: Proceedings of the Fourth International Workshop on Mining Software Repositories*. Washington, DC, USA: IEEE Computer Society, 2007, p. 6.

[13] A. Capiluppi, P. Lago, M. Morisio, and D. e Informatica, "Characteristics of open source projects," *Software Maintenance and Reengineering, 2003. Proceedings. Seventh European Conference on*, pp. 317–327, 2003.

[14] K. Crowston and J. Howison, "The social structure of free and open source software development," *First Monday*, vol. 10, no. 2, 2005.

[15] N. Ducheneaut, "Socialization in an Open Source Software Community: A Socio-Technical Analysis," *Computer Supported Cooperative Work (CSCW)*, vol. 14, no. 4, pp. 323–368, 2005.

[16] M. E. J. Newman, "Finding community structure in networks using the eigenvectors of matrices," *Physical Review E*, vol. 74, no. 3, p. 36104, 2006.

[17] C. Bird, D. Pattison, R. D'Souza, V. Filkov, and P. Devanbu, "Latent social structure in open source projects," in *SIGSOFT '08/FSE-16: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*. New York, NY, USA: ACM, 2008, pp. 24–35.

[18] M. Pinzger, N. Nagappan, and B. Murphy, "Can developer-module networks predict failures?" in *SIGSOFT '08/FSE-16: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*. New York, NY, USA: ACM, 2008, pp. 2–12.

[19] T. Zimmermann and N. Nagappan, "Predicting subsystem failures using dependency graph complexities," in *Proceedings of the The 18th IEEE International Symposium on Software Reliability*, 2007.

[20] ——, "Predicting defects using social network analysis on dependency graphs," in *Proc. of the International Conference on Software Engineering*, 2008.

[21] C. Bird, N. Nagappan, P. Devanbu, H. Gall, and B. Murphy, "Putting it All Together: Using Socio-Technical Networks to Predict Failures," in *Proceedings of the 17th International Symposium on Software Reliability Engineering*, 2009.

[22] E. Carmel, *Global Software Teams: Collaborating Across Borders and Time Zones*. Prentice Hall, 1999.

[23] R. D. Battin, R. Crocker, J. Kreidler, and K. Subramanian, "Leveraging resources in global software development," *IEEE Software*, vol. 18, no. 2, pp. 70–77, March/April 2001.

[24] G. M. Olson and J. S. Olson, "Distance matters," *Human-Computer Interaction*, vol. 15, no. 2/3, pp. 139–178, 2000.

[25] E. Carmel and R. Agarwal, "Tactical approaches for alleviating distance in global software development," *IEEE Software*, vol. 2, no. 18, pp. 22–29, March/April 2001.

[26] J. Herbsleb and R. Grinter, "Architectures, coordination, and distance: Conway's law and beyond," *IEEE Software*, 1999.

[27] J. Herbsleb and A. Mockus, "An empirical study of speed and communication in globally distributed software development," *IEEE Transactions on Software Engineering*, 2003.

[28] N. Rammasubbu and R. Balan, "Globally Distributed Software Development Project Performance: an Empirical Analysis," in *Proceedings SIGSOFT symposium on the foundations of software engineering*, 2007.

[29] C. Bird, N. Nagappan, P. Devanbu, H. Gall, and B. Murphy, "Does distributed development affect software quality?: an empirical case study of windows vista," *Communications of the ACM*, vol. 52, no. 8, pp. 85–93, 2009.

[30] C. Bird and N. Nagappan, "Who? What? Where? Investigating Distributed Development in Open Source Software," Microsoft Research, Tech. Rep., 2011.

[31] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu, "Don't Touch My Code! Examining the Effects of Ownership on Software Quality," in *Proceedings of the the eighth joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*. ACM, 2011.

[32] A. Mockus and J. D. Herbsleb, "Expertise browser: a quantitative approach to identifying expertise," in *Proceedings of the 24th International Conference on Software Engineering*, 2002.

[33] D. W. McDonald and M. S. Ackerman, "Expertise recommender: a flexible recommendation system and architecture," in *Proceedings of the ACM conference on Computer supported cooperative work*, 2000.

[34] T. Fritz, G. Murphy, and E. Hill, "Does a programmer's activity indicate knowledge of code?" in *Proc. of the ACM SIGSOFT symposium on The foundations of software engineering*. ACM, 2007, p. 350.

[35] A. Mockus and D. Weiss, "Predicting risk of software changes," *Bell Labs Technical Journal*, vol. 5, no. 2, pp. 169–180, 2000.