Keynote: Selecting Research Methods for Studying a Participatory Culture in Software Development

Margaret-Anne Storey
Dept. of Computer Science
University of Victoria
Victoria, BC, Canada
mstorey@uvic.ca

ABSTRACT

Recent innovations in social media have led to a paradigm shift in software development, with highly tuned participatory development cultures contributing to crowdsourced content and being supported by media that have become increasingly more social and transparent. Never before in the history of software development have we seen such rapid adoption of new tools by software developers. But there are many unanswered questions about the impact this tool adoption has on the quality of the software, the productivity and skills of the developers, the growth of projects and technologies developers contribute to, and how users can give feedback on and guide the software they use. Answering these questions is not trivial as this participatory development culture has become a virtual network of tightly coupled ecosystems consisting of developers, shared content and media channels.

In our studies, we have found that combining research methods from the social sciences with data mining and software analytics to be the most promising in terms of revealing benefits and challenges from this adoption of social tools. In this talk, I share some of the findings from our studies, discuss the particular research methods we have used, and share our experiences from using those research methods. I also discuss how we as researchers leverage social tools and interact with the participatory development culture to assist with and help us gain feedback on our research. I close with a discussion about how other software engineering researchers could benefit from using social tools and the challenges they may face while doing so.

Categories and Subject Descriptors

H.5.3 [**Group and Organization Interfaces**]: Computer-supported collaborative work

General Terms

Human Factors

EASE '15 April 27 - 29, 2015, Nanjing, China

http://dx.doi.org/10.1145/2745802.2747957

Keywords

Research Methods, Social Media, Software Engineering, Collaboration, Empirical Software Engineering

1. INTRODUCTION

Selecting suitable research methods for empirical software engineering research is ambitious due to the various benefits and challenges each method entails. One must regard the different theoretical stances relating to the nature of the research questions being asked, as well as practical considerations in the application of methods and data collection [6]. Furthermore, studying the human and social factors that occur across distributed virtual or online communities is particularly difficult.

Our research has investigated how development tools and communication media that are infused with social features impact software development communities of practice. Our findings to date indicate that the adoption of *social media* by these communities of developers leads to an emerging *participatory development culture* [21].

In this extended abstract, we describe some of the characteristics of this emergent development culture and the tools that are used to support it. We then discuss the different tradeoffs and challenges faced in selecting and applying research methods for studying development in online communities of practice. Finally, we briefly explore how social tools could foster a more participatory culture in software engineering research and how that may help to accelerate our impact on software engineering practice.

2. EMERGENCE OF A PARTICIPATORY DEVELOPMENT CULTURE

Over the past few decades, software development has transitioned from a predominantly solo activity where developers create small standalone programs, to a widespread distributed exercise where hundreds and even thousands of developers operate within an online community of practice [25] and create complex software ecosystems (e.g., the *Ruby on Rails* project has more than 2500 contributors[1]). This large-scale distributed development effort has been made possible by innovations that include the Internet, the World Wide Web, sophisticated development tools (e.g., integrated development environments, bug trackers, source code repositories), and rich communication channels (e.g., email and

[1] `https://github.com/rails/rails`

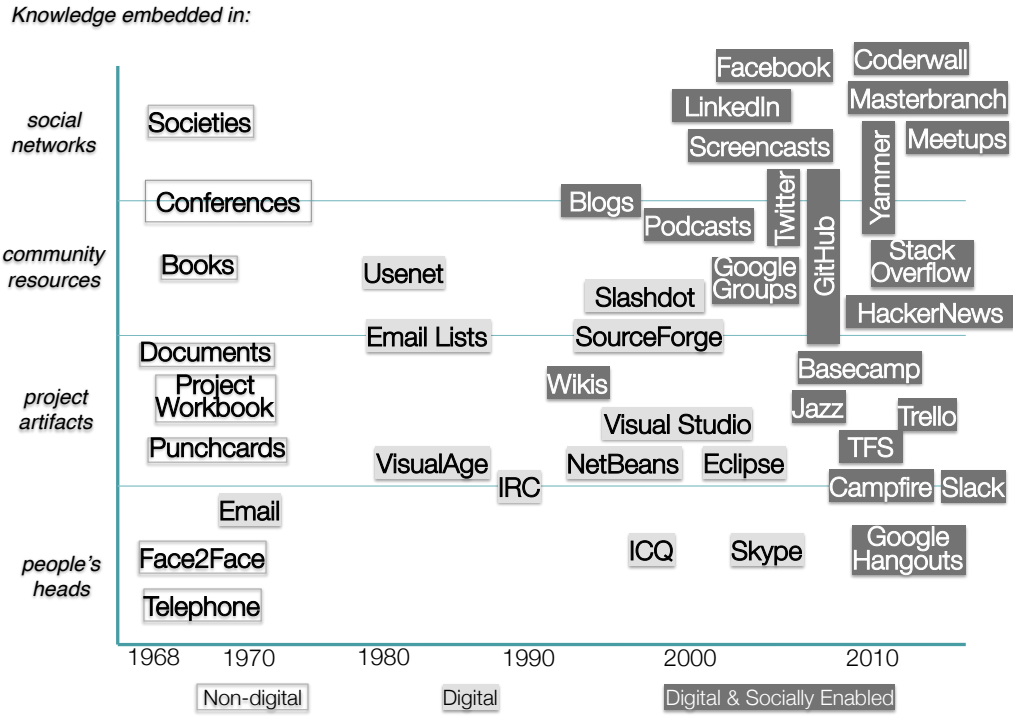

Figure 1: Media channels over time and how they support the transfer of developer knowledge.

online chat tools).

Development and communication tools play a critical role in hosting software, as well as in capturing the history of how the code was developed and documenting how the code can be used or potentially changed. The more recent adoption of social tools, such as social networking and blogging tools, has led to the formation of a *participatory development culture* [21] with lower barriers to entry, strong support for co-creation of artifacts, mentorship opportunities, and appreciation of social relationships [8]. This participatory culture has lead to the emergence of the *social programmer* [22], introduced new means for developers to learn and stay up to date [18], and changed how developers assess themselves and others [17, 11].

We now see an increase in both the number of tools developers use and in the social features these tools provide [21]. These include socially-enabled code hosting tools (e.g., GitHub, BitBucket), management and collaboration task boards (e.g., Trello, ZenHub), focused software development communication tools (e.g., Slack, HipChat), collaborative screen-sharing tools (e.g., Screenhero and Nitrous.io), development activity automation tools (e.g., Hubot), personal and team metric tools (e.g., WakaTime, iDoneThis), and tools to keep developers up to date and help them discover important new technologies (e.g., Hacker News and Twitter).

In Figure 1, we loosely categorize a selection of developer and communication tools according to four kinds of knowledge they help capture or communicate [24, 21]: **knowledge embedded in people's heads** that may be tacit and is best exchanged one-on-one or in small group interactions; **knowledge embedded in development artifacts** that can be accessed directly through a tool; **knowledge stored in a community resource** that is socially generated, maintained and exchanged within emergent communities of practice; and **knowledge about people and social networks**. We see that some communication channels overlap multiple types of knowledge as developers may use many of these tools in flexible ways.

In our recent research, we have been striving to understand and keep up to date with how developer, communication, and social tools support or potentially hinder development within this participatory culture. We studied how developers use, benefit from, and are challenged by tools that include email [15], social networking tools [1], Stack Overflow [2, 14], software tagging features [20, 23], micro-blogging tools [18, 5], social coding environments [4, 26], and screencasting channels [10]. We also conducted a large-scale survey with over 2000 developers to investigate which communication channels developers use for certain activities and the challenges they may face from using a large number of communication channels [21]. Though this a related research, we have observed that tools shape and are shaped by the developers who design and use them.

Never before in the history of software development have we seen such rapid adoption of new tools by software developers. There are many unanswered questions about the impact this tool adoption has on the quality of the soft-

ware, the productivity of the developers and the growth of projects they contribute to, on the users and how they can give feedback on and guide the software they use, or on the skill or technological development of the community as a whole. Answering these questions is not trivial as it's influenced by intertwined social, human and technological issues. In the next section, we discuss the main research methods we have used in our research, and describe the challenges we are facing (and expect to continue facing) in future work.

3. RESEARCH METHODS FOR STUDYING A PARTICIPATORY CULTURE

In our studies of this participatory development culture, our goal has not been to investigate or experiment with new tools or ideas (which would point in the direction of experimental studies), but rather to understand the impact of the social tools developers already adopt on their development practices and on the community. To study existing practices, there are two main approaches to choose from. The first relies on mining and analyzing the *trace data* that developers leave behind when they use development and communication tools. The second approach involves the collection and analysis of *participant data* from observing, surveying and interviewing community members. We discuss these two approaches below and then describe how and why we blend both in our research.

3.1 Mining software repositories

The prevalence of development, communication and social tools has led to an increase in the availability of operational and trace data from programs and developer activities. From program data, we have runtime traces, program logs, system events, failure logs, and performance metrics. From users, we have access to usage logs and user reports, and from development tools, we have different versions, bug data, commit data, and testing results.

The research areas of Mining Software Repositories[2] and Software Analytics[3] develop methods and tools for mining and analyzing repository and trace data with the intent to improve developer productivity and software quality. This research has resulted in some important innovations and insights, such as bug prediction models and recommendation tools.

One advantage of research methods that rely on this data is that the participants make these records with no interference from the experimenters [12] and therefore the participants studied are not subject to *reactivity*. These tools also make it possible to collect data from a wide array of sources, increasing the external validity of the findings [16]. In the cases where data is publicly available, it can be easier for researchers to replicate findings.

A disadvantage is that as more and more tools capture operational data (not from just from developers, but also from users), researchers have to deal with the increasing variety, velocity and volume of this *big data*, making it difficult and expensive to study. But another—and perhaps more important—disadvantage concerns the internal validity of these studies, as such methods may be limited at explaining why or revealing factors that may influence why certain behaviors or phenomena occur [16].

[2] `http://2015.msrconf.org/index.php`

[3] `http://research.microsoft.com/en-us/groups/sa/softwareanalyticsinpractice_minitutorial_icse2012.pdf`

3.2 Borrowing methods from the social sciences

Although data mining and data analytics have been effective at describing or predicting certain behaviors, they are not so effective at explaining why certain behaviors may occur, or how certain development practices or tools could potentially be improved. Trace data may be poorly linked to the concepts we wish to explore or understand [9], concepts such as stakeholder motivations or barriers to community participation. When development data is mined, there is often the underlying assumption that developers are rational *"animals"*, when in fact they may not be at all rational [7] and may have private motivations for their actions.

Furthermore, much of the work that is done in developing software is *invisible work* [19] and cannot be studied by considering the development artifacts alone. Naur points out that viable software is more than just the externalized code and documentation—it is also dependent on extensive *tacit knowledge* that resides in the heads of developers who authored it, maintain it, or use it [13]. And even when development activities or development knowledge may be visible or explicit, this information may be subtly *fragmented* across multiple channels and tools [21]. This fragmentation of developer knowledge and communication is exacerbated as developers create and adopt more and more tools to support their development work. Finally, when studying social media tools, "Dark Matter Developers" that *lurk* or simply use information posted on social channels do not leave a trace behind them; however, our studies need to be aware of such developers' needs and influences[4].

To study these more subtle yet important human and social issues, we have borrowed and adapted research methods (e.g., ethnographies, interviews and surveys) from the human and social sciences [12]. These methods have given us important insights into the benefits social tools bring and the barriers developers may face using development tools and communication channels to participate.

3.3 Bridging the methodological divide

Clearly there are advantages and disadvantages to the choice of research methods [6] and certain tradeoffs need to be made. Siegmund et al. note: *"There is an inherent tradeoff in empirical research: Do we want observations that we can fully explain, but with a limited generalizability, or do we want results that are applicable to a variety of circumstances, but where we cannot reliably explain underlying factors and relationships? Due to the options' different objectives, we cannot choose both."* [16] McGrath also describes tradeoffs in terms of the **precision** of the data that can be collected, the **realism** of the setting studied, and the collected evidence's **generalizability** to the possible population of actors [12].

Our research approach has been to follow a pragmatic ap-

[4] `http://www.hanselman.com/blog/DarkMatterDevelopersTheUnseen99.aspx`

proach by using **mixed methods** [3] where we let our research questions (which may be exploratory, explanatory or confirmatory) guide the selection and order of the appropriate research method or methods [6]. In our studies of communication and social channels, the methods we have used include: mining and analysis of software artifacts, ethnographic observations, and interviews and surveys.

The *role of theory* in our studies varies greatly depending on the stage of our research [6]. In the case when we have an existing (but perhaps preliminary) theory, the theory guides which variables should be measured. In the case where a theory is emerging, we may use it to categorize the data. We have found the need to use multiple methods to assist in theory development.

Too often researchers divide research methods into **qualitative** and **quantitative** camps. Thankfully most research methods involve the collection of both kinds of data, although a method may favor one kind of data over the other. In the case of mining, the data is typically quantitative in nature (but not always, e.g, email communications), and in the case of participant studies in the field, the data may be predominantly qualitative, but again not always. For the large-scale surveys we have conducted (some with over 2000 developers), many of our questions were quantitative (e.g., when we probed on the number of developers they collaborated with).

4. TOWARDS A PARTICIPATORY RESEARCH CULTURE

We have found that social media is causing a paradigm shift in software engineering (just as it has in domains such as politics and journalism), and likewise believe that it may result in changes in our research community.

During the course of our research, we have found it beneficial to use social media to study social media use in software engineering, and to disseminate our research findings. Understanding the culture and nuances of social media language enabled us to reach out to and connect with study participants. In our most recent study, we attracted over 2000 responses to a long and time consuming survey by informing our participants that our results would be openly shared as we have done with other studies—e.g., we tweeted and blogged[5] about the results from our study of Twitter [18] and received excellent feedback that helped us validate our findings and add additional insights to our research results. Our blog saw thousands of views in a single day and dozens of comments from real practitioners. These channels provide a way for us to validate our findings at a speed and scale not possible before. As our collaborator, Gousios recently blogged[6]: *"The benefit is mutual: developers learn about exiting results while our research is getting spread. My blog post about the results of the ICSE 2015 paper is by far the most read one in my blog (around 8k views now). I really doubt that that many people read my paper."* Furthermore, we can engage in a more *in-depth discussion* with participants providing us with a deeper understanding of the phenomena we study as well as revealing new research questions.

Many software engineering researchers have already established excellent social media literacy skills. We have used our social graph to gain research data and insights, and to form alliances with developers and collaborations with other researchers. Just as some software engineers consider Twitter to be an essential tool in keeping up to date on technology developments [18], we also find it a useful avenue for disseminating our research and learning about related research in our field. It also becomes a useful backchannel for discussing research as it is being presented at conferences.

However, there are many challenges and risks from using social tools in our research community. We mentioned the need for researchers to develop improved literacy skills, otherwise they may miss out on research developments. Another risk is that easier access to participants through public channels (such as GitHub) may lead to us inadvertently spamming our participants. We need to take care to reach out to them when we have very good reasons to do so—one way to ensure this is to pilot smaller studies and seek feedback on preliminary findings before reaching out to a wider audience. In our studies, we make sure to explain the purpose of the study and its benefits, and not to send invites to the same people we used for previous studies.

Another issue we need to be aware of is to ensure that we do not somehow cause harm to our participants. We refer to the now famous Facebook study that manipulated users' feeds to study emotional impact: *"Having written and designed this experiment myself, I can tell you that our goal was never to upset anyone. I can understand why some people have concerns about it, and my coauthors and I are very sorry for the way the paper described the research and any anxiety it caused. In hindsight, the research benefits of the paper may not have justified all of this anxiety."* [7]

Lastly, our future work should consider if our interactions with our research participants have an impact on their practices when we share and discuss our results with them. Although, we have not intended it so far, our research may be approaching an *action research* methodology [6].

In conclusion, we suggest that social media can have a transformative impact on software engineering research—through social media, researchers have the opportunity to influence and guide the industry. We further propose that researchers can benefit from the use of social media to help in sharing and disseminating research results with one another and in forming collaborations with others. We look forward to discussing the benefits and challenges of social media use for both developers and researchers in software engineering.

5. ADDITIONAL AUTHORS

Alexey Zagalsky (University of Victoria, BC, Canada `alexeyza@uvic.ca`) and Leif Singer (University of Victoria, BC, Canada `lsinger@uvic.ca`).

[5] `http://blog.leif.me/2013/11/how-software-developers-use-twitter/`

[6] `http://www.gousios.gr/blog/Scaling-qualitative-research/`

[7] `https://www.facebook.com/akramer/posts/10152987150867796`

6. REFERENCES

[1] A. Begel, J. Bosch, and M.-A. Storey. Social networking meets software development: Perspectives from github, msdn, stack exchange, and topcoder. *Software, IEEE*, 30(1):52–66, 2013.

[2] B. Cleary, C. Gomez, M.-A. Storey, L. Singer, and C. Treude. Analyzing the friendliness of exchanges in an online software developer community. In *6th Int. Workshop Cooperative and Human Aspects of Software Engineering (CHASE2013)*, pages 159–160, 2013.

[3] J. W. Creswell. *Research design: Qualitative, quantitative, and mixed methods approaches*. Sage publications, 2013.

[4] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb. Social coding in github: Transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, CSCW '12, pages 1277–1286, New York, NY, USA, 2012. ACM.

[5] K. Dullemond, B. van Gameren, M.-A. Storey, and A. van Deursen. Fixing the "out of sight out of mind" problem: One year of mood-based microblogging in a distributed software team. In *Proc. 10th Working Conf. Mining Software Repositories*, MSR '13, pages 267–276, Piscataway, NJ, USA, 2013. IEEE Press.

[6] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian. Selecting empirical methods for software engineering research. In *Guide to advanced empirical software engineering*, pages 285–311. Springer, 2008.

[7] R. Harper, C. Bird, T. Zimmermann, and B. Murphy. Dwelling in software: Aspects of the felt-life of engineers in large software projects. In *Proceedings of the 13th European Conference on Computer Supported Cooperative Work (ECSCW '13)*. Springer, September 2013.

[8] H. Jenkins. *Confronting the challenges of participatory culture: Media education for the 21st century*. Mit Press, 2009.

[9] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian. The promises and perils of mining github. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 92–101, New York, NY, USA, 2014. ACM.

[10] L. MacLeod, A. Bergen, and M.-A. Storey. Code, camera action: How software developers document and share program knowledge using youtube. In *2015 23rd IEEE International Conference on Program Comprehension (ICPC) (to appear)*, 2015.

[11] J. Marlow, L. Dabbish, and J. Herbsleb. Impression formation in online peer production: Activity traces and personal profiles in github. In *Proc. 2013 Conf. Comput. Supported Cooperative Work*, CSCW '13, pages 117–128, New York, USA, 2013. ACM.

[12] E. Mcgrath. Methodology matters: Doing research in the behavioral and social sciences. In *Readings in Human-Computer Interaction: Toward the Year 2000 (2nd ed.* Citeseer, 1995.

[13] P. Naur. Programming as theory building. *Microprocessing and microprogramming*, 15(5):253–261, 1985.

[14] C. Parnin, C. Treude, L. Grammel, and M.-A. Storey. Crowd documentation: Exploring the coverage and the dynamics of api discussions on stack overflow. Technical Report GIT-CS-12-05, Georgia Tech, 2012.

[15] P. C. Rigby, B. Cleary, F. Painchaud, M.-A. Storey, and D. M. German. Contemporary peer review in action: Lessons from open source development. *IEEE Software*, 29(6):56–61, 2012.

[16] J. Siegmund, N. Siegmund, and S. Apel. Views on internal and external validity in empirical software engineering. In *Proceedings of the 37th International Conference on Software Engineering, ICSE 2015, (to appear)*, 2015.

[17] L. Singer, F. Figueira Filho, B. Cleary, C. Treude, M.-A. Storey, and K. Schneider. Mutual assessment in the social programmer ecosystem: An empirical investigation of developer profile aggregators. In *Proceedings of the 2013 conference on Computer supported cooperative work*, pages 103–116. ACM, 2013.

[18] L. Singer, F. Figueira Filho, and M.-A. Storey. Software engineering at the speed of light: How developers stay current using twitter. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 211–221, New York, NY, USA, 2014. ACM.

[19] S. Star and A. Strauss. Layers of silence, arenas of voice: The ecology of visible and invisible work. *Computer Supported Cooperative Work (CSCW)*, 8(1-2):9–30, 1999.

[20] M.-A. Storey, L.-T. Cheng, I. Bull, and P. C. Rigby. Shared waypoints and social tagging to support collaboration in software development. In *Proc. 2006 20th Anniversary Conf. Comput. Supported Cooperative Work*, CSCW '06, pages 195–198, New York, USA, 2006. ACM.

[21] M.-A. Storey, L. Singer, B. Cleary, F. Figueira Filho, and A. Zagalsky. The (r)evolution of social media in software engineering. In *Proc. of the 36th Intl. Conf. on Software Engineering, Future of Software Engineering*, FOSE 2014, pages 100–116, New York, NY, USA, 2014. ACM.

[22] C. Treude, F. Figueira Filho, B. Cleary, and M.-A. Storey. Programming in a socially networked world: the evolution of the social programmer. *The Future of Collaborative Software Development*, pages 1–3, 2012.

[23] C. Treude and M.-A. Storey. Work item tagging: Communicating concerns in collaborative software development. *Software Engineering, IEEE Transactions*, 38(1):19–34, 2012.

[24] M. M. Wasko and S. Faraj. "It is what one does": why people participate and help others in electronic communities of practice. *The Journal of Strategic Inform. Systems*, 9(2-3):155 – 173, 2000.

[25] E. C. Wenger and W. M. Snyder. Communities of practice: The organizational frontier. *Harvard business review*, 78(1):139–146, 2000.

[26] A. Zagalsky, O. Barzilay, and A. Yehudai. Example overflow: Using social media for code recommendation. In *Proceedings of the Third International Workshop on Recommendation Systems for Software Engineering*, pages 38–42. IEEE Press, 2012.