Thesis Proposal

Software Developers Using Signals in Transparent Environments

Jason Tsay
Institute for Software Research
Carnegie Mellon University
jtsay@cs.cmu.edu

December 11, 2015

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

Thesis Committee

James Herbsleb (Co-chair)
Institute for Software Research
Carnegie Mellon University

Laura Dabbish (Co-chair)
Human-Computer Interaction Institute
Carnegie Mellon University

Claire Le Goues
Institute for Software Research
Carnegie Mellon University

André van der Hoek
Department of Informatics
University of California, Irvine

ABSTRACT

Software development environments are increasingly integrating social media features that allow transparency of software projects and developers by making visible all work-related activity that software developers can use to enhance their work. Developers in these environments often choose to interact with other projects or developers, whether through forming dependencies, learning from an expert, or evaluating code contributions. Transparency can inform these decisions by allowing interested developers to see an entire network of information, such as all the developers that work on a particular project and their prior development history. With this promise of staying aware and interacting with an ecosystem of potentially millions of projects and developers comes the peril of consuming overwhelming amounts of mostly noisy information generated by the broadcast development activity of these projects and developers. However, by identifying what information developers need to be aware of in transparent environments, there is the opportunity to inform new practices and tools that assist developers in their tasks by only displaying information that is most relevant for the current task.

My dissertation work is to identify signals that developers make use of during tasks in transparent development environments and to use this knowledge to create tools that assist developers in performing these tasks. When developers are evaluating contributions, they make use of technical signals from the contribution and social signals from the submitter such as social connections and prior interaction on the project. Developers solving problems through discussion on contributions use signals such as the political influence of the community and how new the submitter is to inform decisions such as the scope of implemented solutions and the necessary amount of etiquette in a response. When deciding to use or participate in a project, developers make inferences about working dynamics, personal utility, and the project's community. I propose the development of a tool that uses signals to assist developers in finding and evaluating projects in transparent development environments. Depending on the user's current task, the tool should visualize different relevant project-related signals to assist the developer in evaluating projects. For example, a user looking to find a project to form a dependency with will see signals for a project's maturity while a user looking to find a project to learn from might see signals for popularity. The development of this tool will occur in 2 phases: 1) enhanced project summary and 2) personalized project search. The enhanced project summary phase augments potential projects with additional signals that assist the user in evaluating and selecting projects. The personalized project search phase uses information about the user and potential projects to select only relevant projects for the particular user to evaluate.

1. INTRODUCTION

With the advent of development environments equipped with social media, the resulting wealth of information about software projects and developers enables transparency of development work. Software engineers are increasingly leveraging the ability of social media to connect with projects and developers as well as rapidly disseminate and consume information about new technology [27]. Software developers are making use of social networking sites such as Twitter to keep track of relevant projects in their domain to potentially use and to find developers to learn from [26]. We also see a trend of software development-oriented services equipped with social media such as GitHub and Stack Overflow where relationships between developers and actions on code are made visible. This social information about software development enables many open source software projects and developers to operate with an unprecedented degree of transparency where outside developers are able to observe in detail the activity of a project's members and vice-versa. By freely observing how projects or developers work, software developers have the opportunity to enhance their own work by choosing to interact with said projects or developers. For example, when a project manager receives a contribution from a newcomer, they choose whether to interact with the developer and their code contribution. In a transparent environment, project managers are able to view all of the prior projects that a newcomer might have participated in and evaluate them as signals of developer skill before deciding whether to accept a contribution or even potentially recruit the developer. Conversely, newcomers looking for open source projects to join may need to decide which projects are worth their time. In a transparent environment, these new developers might investigate what prior contribution attempts look like and how they have fared, as signals of openness and project norms.

The information that enables transparency also comes with the risk of overloading developers with information. The social media systems these transparent environments are based on are also associated with an overwhelming amount of information [26]. Many developers in these systems find it challenging to effectively consume the sheer amount of content, developing ad-hoc strategies to filter and skim. However, whereas missing content on Twitter often has little consequence, missing important work-related information in transparent environments may have negative impacts on productivity or project management. Especially with how visible work is, missing important cues in the environment such as submitted contributions or replying to comments potentially creates negative impressions of the project or developer [7]. This is especially true at "web scale", where developers are potentially wading through information from potentially millions of developers and millions of software repositories when making work decisions. Even if information is overwhelmingly available through these systems about specific projects or developers, they often do not directly answer questions developers have about projects or other developers. For example, a developer evaluating a project as a potential dependency may want to know how mature the project is. Though a project's maturity is not directly visible, a developer might look at various signals the project broadcasts such as recent commit activity, number of versions, and the community size to infer the project's maturity indirectly. Due to the cost of navigating the firehose of development-related information in transparent environment and then additionally interpreting and making inferences, developers may be forced to develop ineffective ad-hoc techniques such as constantly checking feeds [7] or even refusing to participate in social aspects of development.

By understanding how software developers use information made available through transparency, there is the opportunity to identify the most relevant signals for the particular developer to enable connecting or staying aware of projects or other developers without the risk of information overload. While transparency through broadcasting development-related information in software development promises the ability to freely connect with software developers and projects to make better development decisions, the peril of dealing with an overwhelming amount of information from millions of possible developers and projects is also present. My work is to make sense of how developers in these transparent environments currently understand and make use of the information they see. Developers observe the information that other developers and projects broadcast, and then use this information to make inferences that are used to inform development decisions. This inference process is describable using the lens of signaling theory [9,10]. Signals are observable pieces of information that indicate some hidden quality of the person or entity that generated the signal [9]. Transparent environments provide a rich new source of signals about people, communities, and activity. These signals give insight into exactly what information developers use to inform decisions that arise during software development.

My dissertation work is a series of empirical studies that identify signals and inferences developers make in transparent environments which I propose to leverage in a tool that assists developers in finding and evaluating software projects. I identify signals and inferences developers make in various development tasks such as evaluating contributions or projects in a series of empirical studies on the "social coding" open source hosting site GitHub, a very popular contemporary example of a transparent development environment. Informed by how developers use signals identified in prior studies, I propose the development of a tool that assists developers in finding and evaluating open source software projects that dynamically visualizes relevant signals about projects and the developers who work on them depending on the user and their current task.

2. PRIOR WORK

As software development, in particular open source software, often requires coordination between multiple projects and developers, some who may be globally distributed [15], software developers engage in information seeking behaviors to assist in making decisions. In particular, awareness of the activity of projects and its developers is crucial, leading to the development of processes and tools to assist developers in maintaining awareness. However, with the advent of transparent development environments, there is the opportunity to make use of an unprecedented amount of information about projects and developers at the cost of having to learn how to consume large amounts of information. One useful lens for understanding how developers consume information is signaling theory.

2.1. Information Seeking in Open Source Software

Literature on open source software suggests that open source software developers have always needed to seek out information about software projects and other developers to assist in making work-related information. In a study by Scaffidi et al. [25] of what traits of web macro scripts predict reuse, they find that users tended to traits related to authorship, mass appeal, and length (only for self-reuse). These findings suggest that users looking to re-use scripts look for indications that the author has made highly reusable scripts in the past and whether the script in question targets URLs that many other scripts also target. When evaluating code contributions for technical correctness through a peer review process, core members often collect information about the contribution in order to select relevant contributions to evaluate and assist in the evaluation. Rigby et al. [21] found in their examination of different peer review processes in the Apache server open source project that while reviews are broadcast to a wide audience, expert core members would selfselect what contributions to review, based on the specialization or expertise of the core member in question. This broadcast mechanism for assigning reviews also comes with the tradeoff of bothering core members with unnecessary noise while allowing for non-core developers to potentially participate in the conversation. Ko and Chilana [16] found that this broadcast mechanism for discussion around bug reports led to difficulty in finding relevant information such as user feedback for software behavior issues. In particular, divergent conversations without clear measures for quality tended to have a noisy "sea of critiques" from non-core users, leading to decisions being made by authority rather than using information collected by the community.

Developers also seek out social information about software projects to inform their participation. Von Krogh [17] found in his study of the contribution process in the Freenet open source project that successful newcomers must follow "joining scripts" that involve gathering social information about core members of the project before submitting a contribution. These joining scripts involve participating in prior activity such as lurking on the project's mailing list, participating in technical discussions, and reporting bugs. Ducheneaut [11] noted that developers looking to make successful contributions to the Python project needed to learn project norms and identify members of the core project team in order to socialize themselves to the project. In order to successfully start the contribution evaluation process, a submitting developer needed to "recruit" core members of the project as a network of "allies".

2.2. Awareness Processes and Tools

As open source software tends to require distributed software developers to coordinate their efforts [5], open source software developers developed processes to stay aware of their fellow developers through regularly seeking information about their work activities. Gutwin et al. [14] found that developers in open source software projects sought awareness information such as who is working on what part of the project through using simple text communication such as mailing lists, text chat, and commit logs. Projects developed norms of creating short, informative mailing list messages, regularly "overhearing" discussions on lists, and ensuring that important decisions are kept "public". Rigby and Storey [22] found in their study of peer review on open source software projects that developers on a project would use similar work awareness information from the mailing list in order to select code contributions to review. In their study, developers also suffered from "too much awareness" and needed filtering techniques to manage the information overload.

As effectively maintaining awareness is often costly, there are a class of tools that assist developers in maintaining awareness of the activities of software projects and its members. Palantir [23], for example, is an awareness tool that notifies developers when changes are made to relevant work artifacts. Other tools, such as Codebook [1], construct large-scale graph models of linked artifacts, seeking to aid developers in exploring and answering questions about code, documentation, and people. The tool generates the network graph and then a domain expert queries the graph to find information such as who is responsible for a certain product or people connected to certain work artifacts.

2.3. Software Engineering in Transparent Environments

While open source developers use awareness information to coordinate efforts with other developers within a project, development environments integrating social media features provide a wealth of new actionable information regarding outside developers and projects which compounds the risk of overloading developers with noise. Despite the popularity for personal use [2] and application in a few corporate settings [8], we are only beginning to learn about social media's impact on software development. Singer et al. [26] in their study of how open source developers use Twitter found that developers leveraged their networks to both stay aware of software development trends and learn about new technologies. Developers using Twitter faced challenges in how to manage consuming large amounts of information through developing strategies such as filtering tweets and curating their following networks.

Along with normal social media usage, software developers are also making use of development environments that integrate social media features such as GitHub [13]. These environments integrate social media features by allowing developers and projects to broadcast their activity to those who are interested. This allows transparency to the work actions of these developers and projects [6] at the potential cost of having to consume and manage large amounts of information [26]. Previous qualitative research on GitHub suggests that how developers manage this information made visible via transparency is by selectively using pieces of information to make inferences about other developers and projects. Dabbish et al. [7] found that open source software developers use these inferences in practical ways, for instance to help manage external contributions to their projects, discover project user needs, and recruit developers. This research also suggested that the feed of project activity generated by social media strongly influences who and what developers attend to. Developers in this study used signals of community attention to a project or event in the feed to determine if a project was worth using or a discussion was worth reading. Marlow et al. [19] found that GitHub developers used information visible in the environment in order to form impressions of users and projects. Pham et al. [20] found that project managers used many different factors when assessing how much testing a contribution requires, such as the size of the change, the type of contribution, and how much they trusted the submitter. Some of these factors are signals that project managers used to assess quality and risk of the contribution.

2.4. Signaling Theory and Social Networks

Signaling theory is a useful lens for understanding how information broadcast in social media is used in transparent development environments. Signals are observable pieces of information that are used to infer hidden qualities of the signaler [9]. Signals vary in their costliness to produce and thus their strength of association with the signaled quality. "Assessment" signals are reliable signals where the indicated quality is inherent to generating the signal. Writing correct code for complex functionality is a reliable signal for coding skill, for example. "Conventional" signals, on the other hand, are potentially unreliable signals that indicate a quality simply by agreement among members of a community. For example, a resume listing involvement in a number of impressive-sounding projects may or may not reflect meaningful experience and skill. Conventional signals are meaningful only if a community has enforcement norms and mechanisms that keep people honest.

To use the previous example, embellishing a resume is frowned upon and carries the risk of a negative reputation. Donath [10] finds rich patterns of signaling and deception in online communities to infer member identities. Social media tied to software project activity provide signals about underlying project properties such as quality, collaborative environment, and member commitment.

3. **THESIS:** SOFTWARE DEVELOPERS USING SIGNALS IN TRANSPARENT ENVIRONMENTS

As seen in the reviewed literature, open source software developers have always needed to engage in information-seeking behaviors to assist in tasks such as evaluating code submissions for a project. With the advent of transparent development environments comes new possibilities for developers to collect and use information at larger scales, such as developers curating their social networks on Twitter to stay aware of relevant development trends or GitHub users monitoring their activity feed of potentially hundreds of developers and projects to find interesting events or projects. Transparent environments also bring an overwhelming amount of possible projects and developers to stay aware of. Awareness practices and tools for traditional open source software development may no longer scale to these transparent environments, as they tend to be focused on developers navigating single projects rather than creating connections with multiple projects and their developers as is common in transparent environments.

However, by identifying what information developers need to be aware of in transparent environments, there is the opportunity to inform new practices and tools to assist developers. Prior work on transparent environments such as GitHub suggests that developers are using specific pieces of information as signals to make inferences about projects and developers. An understanding of these signals along with the types of inferences made during different development tasks such as evaluating code or recruiting developers would allow for the creation of subsets of highly relevant information for a specific developer's current task. These subsets of information could then be presented directly to the developer rather than the current method of having the developer search manually or using ad-hoc techniques such as relying on serendipity.

Thesis Statement:

Identifying the signals that software developers use in transparent development environments to form inferences about other developers and projects informs the creation of practices and tools that assist developers in making faster and more accurate decisions.

To support this claim, my dissertation work is to identify signals and inferences through empirical studies that developers in current transparent development environments such as GitHub make use of in tasks such as evaluating contributions or software projects. Based on these signals, I propose the development of a tool that assists developers in finding relevant projects by leveraging project-related signals.

4. PRELIMINARY WORK

My preliminary work is a series of empirical studies of how developers in social media-equipped transparent development environments such as GitHub use information available to them about other developers and projects as signals to inform their own development decisions.

4.1. Signals for Evaluating Contributions in GitHub

With the advent of social media and distributed version control systems, many open source software projects operate with an unprecedented degree of transparency. This transparency enables developers to better use information such as technical value and social connections as signals when making work decisions. However with information available from potentially millions of developers and millions of repositories in these transparent work environments, then what information do software developers use when evaluating software contributions? Conventional wisdom on open source projects suggests that technical merit of the contribution itself should be all-important [24]. Prior literature on traditional open source suggests that prior interactions with a project and project culture should also have an important effect. In a transparent environment, the visible relationships between users may also have an important effect on contribution decisions. This study [28] aimed at better understanding how these different signals are used by software project managers in GitHub to evaluate pull requests, which are one of the primary methods for contributing code in GitHub. To determine which signals project managers used, this study analyzed the association of various technical and social measures with the likelihood of contribution acceptance.

4.1.1. Quantitative Study of Factors Predicting Pull Request Acceptance

This work was a quantitative study of pull requests from thousands of GitHub projects for what social and technical factors are associated with contribution acceptance. The technical and social factors used in the study were based on potentially important aspects of contributing to open source software and online communities identified in prior literature. These social and technical contribution factors became the following hypotheses:

H1: Contributions that show signs of following technical contribution norms are more likely to be accepted.

H2: Contributions from submitters with a stronger social connection to the project are more likely to be accepted.

H3: Contributions with a high amount of discussion are less likely to be accepted.

H4: Acceptance of highly discussed contributions will be moderated by both social and technical factors.

H5: Contributions from submitters with a high status in the general community are more likely to be accepted.

H6: Contributions from submitters that hold higher status in a specific project are more likely to be accepted.

H7: Contributions to established projects are less likely to be accepted.

The dataset used in the study was a set of 659,501 closed pull requests from 12,482 GitHub projects. Projects were sampled for active, collaborative projects with at least one event of activity within the past week prior to collection, three unique contributors, and at least one closed pull request. From this dataset, a set of measures operationalizing the contribution factors to be tested were created on three levels: per pull request, per user, and per project. The dataset was multi-level in nature as users submit multiple pull requests and multiple users participate in projects. The outcome measure for our statistical model was pull request acceptance. The analysis was a multi-level mixed effects logistic regression model due to the dichotomous outcome variable and nested levels in the dataset.

4.1.2. Key Findings

This study found that project managers made use of information signaling both good technical contribution practices (H1) for a pull request and the strength of the social connection (H2) between the submitter and project manager when evaluating pull requests. Specifically, both good technical practices such as including test cases and being strongly connected to the project manager such as through GitHub's following mechanism were both associated with increased likelihood of pull request acceptance, especially social connection which had the strongest influence on acceptance likelihood in the model. Pull requests with many comments were much less likely to be accepted (H3), moderated by the submitter's prior interaction in the project (H4). This finding may suggest that highly discussed contributions were risky or uncertain in some way, unless the submitter has much prior experience in the project. Submitters with high status in the community through having many followers (H5) or in the project through collaborator status (H6) also were more likely to have their contributions accepted. Well-established projects were more conservative in accepting pull requests (H7), with older or more popular projects being less likely to accept pull requests. These findings provide evidence that developers use both technical and social information when evaluating potential contributions to open source software projects.

4.1.3. Challenges Identified

While both technical contribution norms and social connections were associated with pull request acceptance, our measures for social contribution had much stronger associations than our technical contribution norm measures. One possible explanation is that when project managers are evaluating pull requests, when the technical evaluation costs are high, they may decide to use social signals to lessen the rigor of the evaluation due to trust or familiarity which may reduce the chance that flaws are found. If this is the case, then a challenge this study identifies for developers interpreting information in their environment is assessment cost. As time and resources are always limited, developers also factor the assessment cost in interpreting signals in their tasks. In this example, easier to assess social signals are used in lieu of a more costly technical evaluation. The finding that prior interaction moderates how highly discussed pull requests are evaluated also suggests that the identity of the submitter may change the interpretation of signals used by the project manager. This may suggest a challenge for developers in that interpreting signals is also highly dynamic, perhaps changing depending on the current task or which developers are involved in the task.

4.2. Discussing Problems and Social and Political Influence

Open source software projects often rely on code contributions from a wide variety of developers to extend the capabilities of their software. Project members evaluate these contributions and often engage in extended discussions to decide whether to integrate changes. These discussions have important implications for project management regarding new contributors and evolution of project requirements and direction. Unique to open settings such as open source projects, project members have little or no formal authority over contributors and vice versa [18]. The methods and means they use to attempt to increase their likelihood of acceptance may reveal what constitutes power in this setting. Given the open nature of the environment, there is no prescribed method by which contributions are resolved. What do the outcomes of these lengthy discussions look like? This study [29] examined the content and form of comments in lengthy discussions in response to code contributions in order to extend understanding of the dynamics of collaboration in an open environment. The problems and issues identified in contributions revealed unique challenges of collaborating with a heterogeneous diffuse set of contributors with limited shared understandings and common goals.

4.2.1. Qualitative Study of Contribution Discussions

This work was a qualitative study on lengthy discussions around contributions in GitHub projects. The decision to focus on extended discussions was made as these discussions tend to reveal important social project dynamics and project members' values and mental processes as they articulate arguments for or against a particular change. On the other hand, the average GitHub contribution generates little if any discussion and involve very few developers [13]. From examining literature on discussion around contributions in online environments, a number of research questions were developed with the purpose of advancing our knowledge of how software developers discuss contributions in open environments.

RQ1: What are the different kinds of issues raised around code contributions?

RQ2: How do participants try to influence the decision process in code contributions?

RQ3: What are the different outcomes for proposed code contributions?

RQ4: Is discussion different when the submitter has prior experience with a project?

The dataset for this study consisted of both a set of 423 comments from 115 developers, embedded in extended pull request discussions. This dataset was supplemented with interviews of 47 users of GitHub. Using a grounded approach we focused on the phenomenon of extended contribution discussions (rather than interactions around contributions more broadly) because these discussions are an important aspect of the open collaboration process and a key place where core and peripheral members negotiate around aspects of project evolution and direction [3].

4.2.2. Key Findings

This study explored the kinds of issues core developers raised and the arguments they have over both the appropriateness of the problem that submitters attempted to solve, and the correctness of the implemented solution in a submitted code contribution (RQ1). In the former case, core developers would place the burden of proving that the contribution had actual value for the project on the submitter. In the latter case, core developers questioned how accurate or optimal the solution implemented in the contribution is. Sometimes, third party developers would use this as an opportunity to present their own alternative solutions, even advertising their own pull requests within a pull request.

Due to the open nature of the software projects, other stakeholders from outside the project observed and participated in these extended discussions, sometimes attempting to influence the outcome of the contribution through political means (RQ2). These political maneuvers included rallying support of the audience through "+1" comments or leveraging project or company communities ("I need this change because my company uses your project").

Regarding contribution outcomes, we found that non-member contributions were more likely to be rejected following a long discussion. However, although core teams rejected new submitter's contributions more often, they almost always satisfied the submitter's technical goal by implementing an alternative solution (RQ3).

For new submitters, core members interacted more politely and in cases of conflict, were more likely to implement alternative solutions for these newcomer submitters rather than simply suggest them for the submitter to implement themselves (RQ4).

4.2.3. Challenges Identified

One challenge for both submitters and core members this study identified is the role of third-party stakeholders in the discussion. The transparent nature of GitHub projects allows for these developers to easily participate and even influence the discussion around evaluating contributions. In some cases, these third-party developers even offered competing alternative solutions. This raises a challenge for core members in the extra cost for them to evaluate the contribution. For submitters, it becomes necessary to be aware of this newfound competition and in some cases, even having to justify implementation decisions to both core members and these third party developers.

4.3. Signals for Evaluating Projects

Social media is being integrated into software development environments, allowing software projects to broadcast information about their activities quickly through a large social network. This transparency of work found in environments like GitHub allow software developers to use information about projects as signals to make inferences about software projects to make highly informed choices about which projects to incorporate or participate in. This mixed methods study

[30] first explored what signals developers are making use of when deciding whether to use or participate in a project. The study then verified these findings with a larger sample of developers and projects.

4.3.1. Mixed Methods Study of Signals Used for Tasks

This study used an exploratory sequential mixed methods approach [4] where the first phase is an qualitative exploratory interview study and the second phase is a quantitative validation study that uses a larger sample of developers and projects to validate findings from the first phase. From a review of previous research on awareness in open source software and social software development in environments such as GitHub, we identify the following research questions:

RQ1: What kinds of inferences do developers make when using signals to perform project-related tasks?

RQ2: How do signals and inferences based on them differ depending on the developer's current task?

RQ3: Can simple metrics provide approximate measures of strength for each signal?

The initial qualitative phase of the study explored what signals software developers use when performing project-related tasks through an interview study with 47 developers on GitHub, with a mix of both peripheral and heavy users. The interviews were analyzed using a grounded approach, identifying instances where developers performed tasks of either deciding to use or participate in a project.

The second quantitative phase of the study used signals identified by interviewed developers to develop a set of metrics, corresponding to each type of inference for each type of task. This phase of the study validated the qualitative results by seeing if the statistical associations they suggest were actually present in a different population of GitHub users. The dataset was created from 1862 Ruby gem projects on GitHub, Ruby gems were chosen as a convenient method to get upstream and downstream dependency data, a useful metric for signals identified. Two statistical models were created, one for Using and one for Contributing, each with separate outcome measures representing each task (number of downstream dependencies and number of pull requests respectively). These models used negative binomial regression to associate signal measures with outcomes.

4.3.2. Key Findings

The first phase of the study found in interviews with GitHub developers that they use signals to make inferences about the project's working dynamics, identify personal utility, and evaluate the community around the project. These inference types were identified across both tasks of using and participating in projects (R1) but the signals used and the specific inference differs depending on the task (R2). Project Working Dynamics are inferences about the project's working style and direction, where developers deciding to use a project would identify how active the project is and developers deciding to participate in a project would identify the responsiveness of the core team to contributions. Personal Utility inferences estimate the potential benefit or cost of involvement in the project. For the task of using, developers would infer the project's ease-of-use whereas for the task of participating, developers would infer how accessible the project is to contributions. Community Evaluation inferences determine the degree of community support or participation in the project. For developers looking to use a project, they would infer the degree of community interest or involvement whereas developers deciding to participate in a project would infer the potential impact of their contribution to the community.

In the second phase of the study, we found in our quantitative validation study that developers evaluating projects for usage looked for signals that indicated active projects in that projects with higher commit volume were more likely to get more use. Mature projects were also more likely to be used, as having more project versions or less code churn increased likelihood of usage. Developers also valued projects that were easy to use projects in that projects with less required downstream dependencies were more likely to be used. Developers making the decision of contributing to projects used signals about a project's responsiveness in that the faster a core member closed a pull request (regardless of the outcome), the more likely the project would receive more contributions. Developers also valued projects that were more accessible to contributions in that projects with more issues and versions were more likely to receive contributions.

4.3.3. Challenges Identified

The main challenge identified in this study was that the developer's task greatly changed what signals they looked for and the interpretation. Although there were general types of inferences identified, the signals developers seemed to use varied greatly in type. In fact, to decide whether to use a project, developers seemed to look at commit history, release history, and required dependencies whereas for deciding to participate, developers looked at past pull requests, issues, and release history. This potentially raises the assessment cost of interpreting signals for developers, especially as modern environments are not designed for multiple tasks in mind.

5. PROPOSED WORK

Based on prior empirical studies of how developers use signals in social software development environments, I am developing a tool that assists developers in finding and selecting open source software projects through the application and visualization of relevant project-related signals. This work will occur in two phases: 1) enhanced project summaries that dynamically visualize project signals dependent on the user's desired task 2) personalized project search that uses information about the user and projects to filter and select more relevant projects.

5.1. Open Source Software Project Search and Evaluation Tool

A common yet potentially difficult task almost every software developer faces is searching for relevant software projects to fulfill certain needs. Commonly, developers search for projects to add as dependencies in their own projects, such as useful libraries or plugins. Besides establishing dependencies, developers may also search for projects to stay aware of new or popular technology

[7]. In some cases, developers may search for projects as educational examples or opportunities to make contributions [7].

Despite the ubiquity of this task, there is little support for searching for projects, leading developers to develop their own ad hoc strategies. Developers may leverage their social networks, observing which projects similar peers or experts make use of [26] or perhaps even directly asking other developers. Technological support is largely limited to simple keyword searches, such as GitHub's project search or "Googling it". However, identifying and making use of signals emitted by software projects should assist in both finding and evaluating relevant projects depending on the particular developer's needs.

5.1.1. Tool Architecture

Currently, this tool is developed as a web application with four main components: 1) data collection and aggregation into signals from GitHub and other sources, 2) user profile and task customization, 3) personalized project search, 4) enhanced project summary. These components and how they relate are summarized in Figure 1. This tool is currently written in Ruby and relies on the Ruby on Rails web application framework. Ruby was chosen for the convenience of using the official GitHub API library octokit.rb as well as Ruby projects being historically popular on GitHub.

The core backend activity of this tool is to collect user and project data from sources such as the GitHub API, clean and aggregate such data into actionable signals, and cache the aggregated signals to avoid re-calculation. To collect data, this tool uses a pull model where whenever new projects or developers are encountered, data are pulled from the GitHub API and other sources such as Twitter, Google, and Stack Overflow. In the future, perhaps to deal with scalability or performance issues, alternative data collection approaches may be considered, such as using the GHTorrent dataset [12] to reduce dependency on the GitHub API. After data on projects and users are collected, the tool aggregates the data into actionable signals. For example, all of a project's closed pull requests may be aggregated for the average time to close as a signal of a project's responsiveness to contributions. As data collection and aggregation is computationally expensive and time-consuming, this tool also needs to cache computed signals for future use into a MySQL database. Over time, the state of developers and projects should change, invalidating these cached computed signals and requiring recalculation. Currently, recalculation is planned to occur if the time between last data collection by the tool and last activity of the project or developer is greater than a certain time threshold, perhaps one week.

As the tool makes use of information about the current user, it also requires a mechanism for authenticating a user and creating a profile in order to customize search results and visualizations. For convenience, this tool makes use of the OAuth protocol using a user's GitHub account. This has the advantage of associating customization data to a specific GitHub account as well as lessening security concerns. Due to the OAuth protocol, a user's GitHub password or other sensitive private data is not stored in this tool's database. Along with using data about the current user to personalize results and signals shown, the user's profile in the tool also allows defining what the current user's task is. This task represents a user's motivation for finding projects: finding a project to Use in their own work, discovering projects in relevant domains to Learn from, or evaluating potential projects to Participate in. Based on challenges identified in preliminary studies, the signals used and their interpretation seem to change depending on their task. This suggests that the user's selection of the current task should also dynamically adjust the selection and visualization of projects. The three tasks available to the user may change as necessary depending on user feedback.

The core frontend activity of this tool is the selection and display of open source software projects for a given user. Using project and developer signals from collected data and customized by the current user's profile, the tool returns a set of relevant projects to the user along with a visual summary of signals to help the user select the most fitting project for their current task. These user-facing activities are broken into two components: 1) personalized project search that returns a filtered set of projects based on the current user and 2) enhanced project summary that visualizes project-related signals depending on the current user's task. Details on the enhanced project summary and personalized project search are given in sections 5.2 and 5.3.

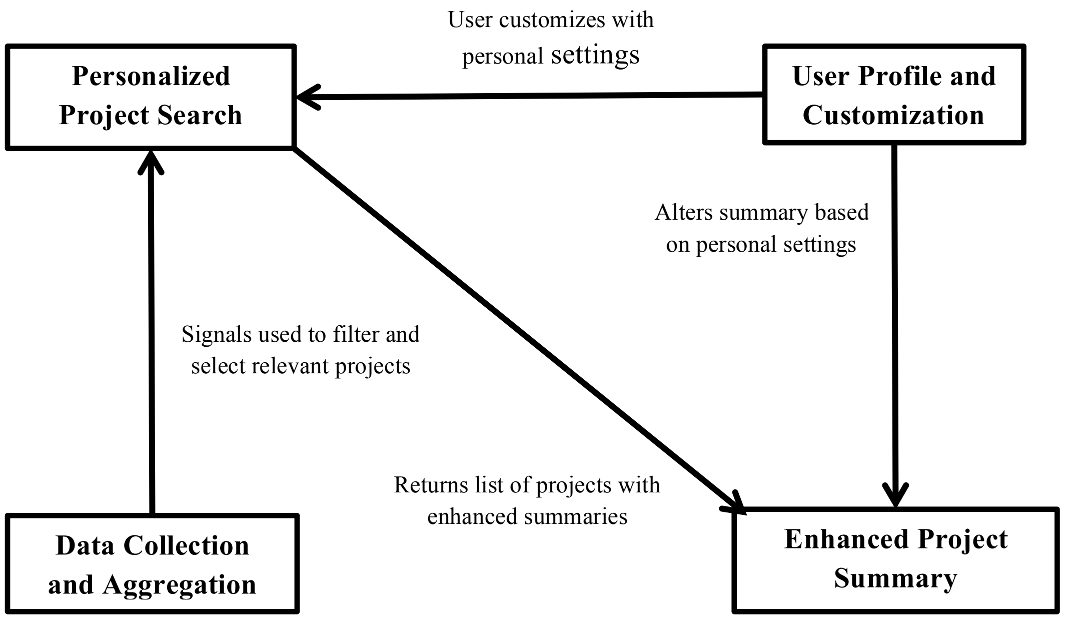

User customizes with personal settings
Personalized Project Search
User Profile and Customization
Alters summary based on personal settings
Signals used to filter and select relevant projects
Returns list of projects with enhanced summaries
Data Collection and Aggregation
Enhanced Project Summary

Figure 1. Architecture diagram for major components of project search tool.

5.1.2. Iterative Implementation Method

My proposed approach for developing this tool is to follow an iterative development method. Inspired by agile methodologies and startup-style development, I will iterate often between development and pilot studies. The goal in using this methodology is to guide the development process using feedback from users in a form of participatory design. Rapid iteration also allows for testing the effectiveness of different features at a minimal cost of implementation effort or time. For example, alternative approaches to visualizing commit velocity could be elicited from the pilot study participant through even paper or whiteboard prototypes. Pilot study participants will be required to have used open source software and GitHub before. Experience searching for open source software projects will be preferred but not required. Most participants will probably be local undergraduate and graduate computer science students. Pilot study sessions will be short, informal think-aloud sessions where the participant is given a hypothetical task and asked to give feedback. Pilot studies are expected to take around 10-15 minutes. The task and goal of each pilot study will change per session depending on features implemented beforehand.

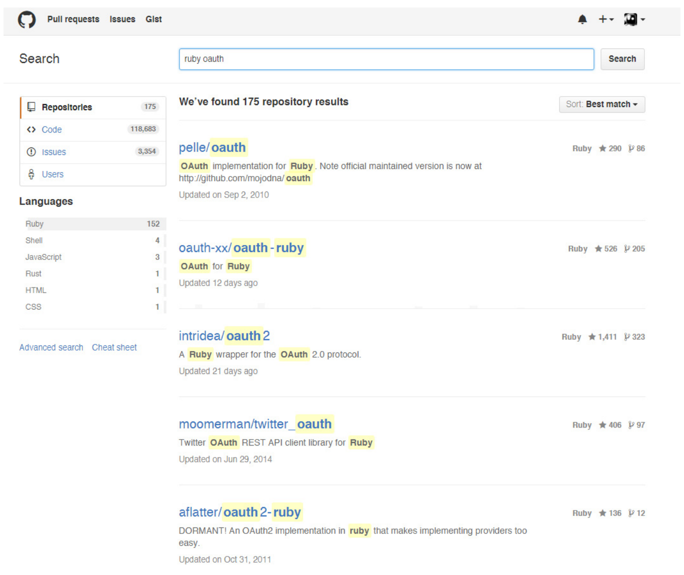
5.1.3. Implementation Phases

This project will occur in two phases: 1) enhanced project summary and 2) personalized project search. I split the user-facing activities into two separate projects in order to both make the overall research project and non-user-facing components more manageable. The first phase of the project, the enhanced project summary, also includes implementation of most of the data collection and aggregation backend and user profiles and customization. The second phase of the project, the personalized project search, will include the extra effort of integration of both phases and the evaluation of the overall tool. The next two subsections detail the implementation of each phase and the evaluation plan for the tool itself.

5.2. Phase 1: Enhanced Project Summary

As developers are using signals from potential software projects to inform evaluation and selection, the first phase of the tool is to assist the developer in effectively finding and selecting projects by displaying a set of relevant signals. Although there are signals that developers already use when finding projects, some of these signals are either costly to gather or are potentially misleading. In the case of GitHub, many of the currently useable signals are primarily for the selection of projects for forming a dependency, other tasks are not as well supported. This tool has the potential to quickly show signals that would be difficult to determine manually and visualize relevant signals depending on the current user's task.

This phase of the tool will focus on enhancing information shown about software projects for the purposes of selection and evaluation based on the current user's task. Using a search engine as a metaphor, this phase of the project simply enhances the information shown about each search result which corresponds to an open source project. For this phase, the tool will not implement any filtering or intelligence for the project search, instead opting to simply return the results of GitHub's existing keyword-based project search engine. Currently, GitHub's project search returns basic descriptive information such as project name and owner but also popularity signals such as number of stars and forks, as seen in Figure 2. This tool will enhance the project summary with additional signals to help developers select the most relevant project for their needs. These signals shown will change depending on the user's current task, to be defined in the user's profile page. For example, a user that wishes to find a project to incorporate into their own software project may wish to see signals of maturity whereas a user looking for projects to participate in may be more interested in signals of responsiveness to contributions.



We've found 175 repository results

pelle/oauth — Ruby ★ 290 86
OAuth implementation for Ruby. Note official maintained version is now at http://github.com/mojodna/oauth
Updated on Sep 2, 2010

oauth-xx/oauth-ruby — Ruby ★ 526 205
OAuth for Ruby
Updated 12 days ago

intridea/oauth2 — Ruby ★ 1,411 323
A Ruby wrapper for the OAuth 2.0 protocol.
Updated 21 days ago

moomerman/twitter_oauth — Ruby ★ 406 97
Twitter OAuth REST API client library for Ruby
Updated on Jun 29, 2014

aflatter/oauth2-ruby — Ruby ★ 136 12
DORMANT! An OAuth2 implementation in ruby that makes implementing providers too easy.
Updated on Oct 31, 2011

Figure 2. Screenshot of existing GitHub search.

5.2.1. Proposed Features

For the enhanced project summary, there are four main features that are required: 1) data collection and aggregation backend, 2) user profile, 3) project search, and 4) enhanced project summary. The proposed user activity flow is as follows: the user first authenticates themselves to the tool using their GitHub account. After the user is logged in, they have the opportunity to define their current task, finding projects for the purposes of: Using, Learning, or Participating. Tasks may be added or removed as necessary depending on pilot study feedback. The user then uses a search bar to find projects via keyword search. For this phase of the project, the project search simply returns results from GitHub's keyword-based project search engine. The results returned from this keyword search are displayed to the user where each potential project contains additional information depending on the current user's task. To examine projects in detail, a callout window will be available to allow users to view additional signals without cluttering the results view. Figure 3 is a mockup of the enhanced project summary.

The main feature of this phase of the project is the visualization of a set of signals to help the developer easily select the most relevant project. Depending on the user's current task, a different set of signals should be displayed for the user. For the task of finding a project to Use, based on earlier studies, the signals displayed should reflect aspects of the working dynamics of the project as well as the cost and benefit of selecting this project. These signals may include project liveliness signals such as commit volume, maturity signals such as project versions and recent code churn, and ease-of-use signals such as number of required upstream dependencies. For the task of finding a project to Participate in, similar signals for working dynamics and the cost and benefit of participation may be used, such as contribution responsiveness signals in the form of average time to close and community impact signals such as number of project versions and issues. For finding a project to learn from, signals that evaluate the project's community may be used, such as social influence signals in the form of the followers of its core members and usage signals in the form of already existing downstream dependencies.

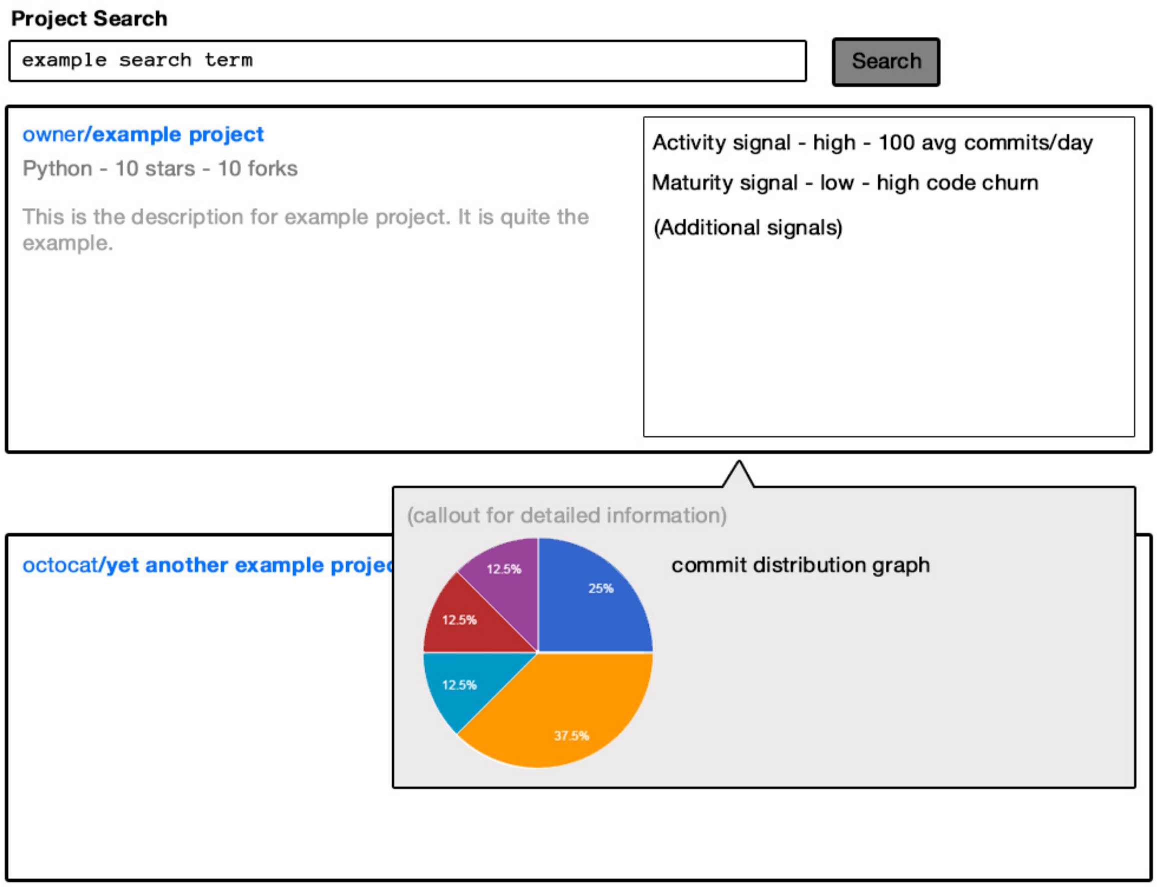

Project Search
example search term
Search
owner/example project
Python - 10 stars - 10 forks
This is the description for example project. It is quite the example.
Activity signal - high - 100 avg commits/day
Maturity signal - low - high code churn
(Additional signals)
(callout for detailed information)
octocat/yet another example projec
commit distribution graph
12.5%
25%
12.5%
12.5%
37.5%

Figure 3. Mockup of enhanced project summary.

5.2.2. Implementation Plan

This phase of the project will need to implement the enhanced project summary UI features along with the backend infrastructure necessary to compute project and user-related signals. As this project will follow an iterative approach, the first step is to implement a minimal working version of this tool quickly in order to start pilot testing. This initial minimal implementation of this project will implement GitHub's keyword search box and include three additional signals that are not relevant to a specific task: recent commit churn (project maturity), number of open/closed pull requests (outside contributions), and average time to close pull requests (responsiveness). This version will also implement the backend infrastructure necessary to collect data about software projects in the search results from the GitHub API, compute these three signals, and cache the computed signals into the tool's database.

The other necessary component for the tool not implemented for the initial minimal implementation is the user authentication and profile. As described earlier, user authentication will be handled using the user's GitHub account via OAuth. This allows the tool to store customization information per GitHub account. For this phase, the main customization will be the definition of what the current user's main task is: Using, Learning from, or Participating in projects.

The main challenge of this phase is the selection and visualization of a set of actionable signals for the user. To avoid information overload, it is necessary to avoid cluttering the project summary with all possible signals hence the decision to dynamically display different signals depending on the current task. At the same time, the method of visualizing specific signals or even projects in general is an important challenge. What is the most efficient method of conveying signals such as commit activity? Politeness in responses from core members? Social distance? Whereas some signals may be simply expressed in numbers, other signals may be more effectively displayed as plots or social networks. Similarly, there is an opportunity to display projects in a more efficient manner than a simple list as most search engine results do. For example, if the user's current task is to find projects to incorporate into their own, that implies the user is comparing and evaluating similar projects against each other. Rather than simply listing projects, a side-by-side comparison view may be more appropriate for this task. The iterative nature of this project may help in selecting signals and visualizations, allowing for rapid experimentation in different sets of signals displayed and visualization methods through pilot studies.

5.3. Phase 2: Personalized Project Search

While developers commonly search for projects for varying reasons, the lack of tools that support project search cause developers to develop ad hoc techniques or rely on serendipity. This phase of the project focuses on improving how developers search for relevant projects by making use of user and project signals to personalize how projects are selected and filtered. Current tools rely on little more than basic keyword searching or Google PageRank-like influence algorithms, this tool has the potential to intelligently match potential software projects to users depending on their current needs, social network, and past activity.

This phase of the tool will focus on using user and project signals to improve on which projects are selected when a user is searching for projects. To again use the metaphor of a search engine, whereas the last phase enhanced the information displayed for each search result, this phase will improve the search results themselves, perhaps even how the search results are displayed to the user as opposed to the ubiquitous simple list that most search engines use. Currently, GitHub's project search returns a list of projects that match keywords and search options using an opaque "best match" algorithm that seems to use some combination of creation date, number of star gazers and forks, and update date as seen in Figure 2. Rather than using such an algorithm, this tool can use data from the current user to assist the search process. For example, if the current user is attempting to find projects to form dependencies with, older, more mature projects should have a higher priority.

5.3.1. Proposed Features

As much of the infrastructure for this project was implemented in the previous phase, this phase of the project will focus on using user and project signals to assist in selecting relevant projects and how to best visualize the result set of projects to the user. The activity flow is the same as described in section 5.2.1 except that the project search and how the results are presented will be personalized to the user.

As with the previous phase, the current user's task of finding projects to Use, Learn from, or Participate in should modify which signals are used when matching and filtering projects to the user. In this phase, rather than solely focusing on what project signals to display to the user, both information about the user and the project are considered. For example, a user looking to incorporate a project into their own may not only want to know the project's maturity but also if projects similar to their own are using the project in question. Users finding potential projects to contribute to may wish to see projects that are more welcoming to newcomers should the current user also be an inexperienced developer.

There is also the opportunity to modify how search results are presented to the user, as opposed to the normal priority list. For users looking to find projects to use, the interface should prioritize comparing projects to each other. On the other hand, if a user is looking for projects to learn from, an interface that visualizes the wider ecosystem of projects and who uses each project may be more appropriate.

5.3.2. Implementation Plan

This phase will implement both personalized project searches and visualizations for resulting projects. Most of the necessary infrastructure should already be implemented in phase 1 except that the backend will need to also collect signals about the current user and the user's working activities such as projects. Similarly to the project signals, these computed user signals will need to be cached in the tool's database. Similar to the first phase, the first step is to implement a minimal working version of personalized project search for pilot testing. This minimal prototype will include everything from phase 1 along with an improved search for users looking to find projects to use. This minimal version should include a side-by-side comparison of potential projects along with filters that prioritize projects depending on whether the developers in the current user's social network also use the project.

The main challenge for this phase is to select a reasonably relevant set of projects using the user's personal data. Whereas prior studies have examined which signals are potentially useful to display to developers, selecting open source projects is a relatively unexplored activity. This phase may become much more exploratory as a result, requiring many iterations with pilot study participants to find a matching algorithm and result display method that satisfies most users. It is an open question for which aspects of a user's profile, activities, and tasks are useful when searching for projects. If a user is looking to find projects to learn from, should the visualization change if the user is new to GitHub? Should suggested projects be similar to projects the user already uses? Or intentionally different to avoid the “filter bubble” problem? If the user is looking for projects to make use of, would they rather look for similar projects instead?

5.4. Tool Evaluation Plan

At the end of both phases and the necessary integration work, it will be necessary to evaluate the tool for its effectiveness in assisting a developer search for open source software projects to use, learn from, or participate in. When the tool is completed, a think-aloud study will be performed that compares this tool against the current state-of-the-art for searching for projects (Google and GitHub search). Participants will be stratified for experience in open source software and general software development in order to include both newcomer and veteran developers. The current evaluation plan is to prepare hypothetical project selection tasks for the user, such as “You are developing a Rails web application and need an OAuth library in Ruby, please find one.” For each task there are two conditions: using existing tools such as Google/GitHub search and using this tool. Participants will be asked to think aloud during the task about the search project and what signals they are using in order to evaluate and eventually select projects. The tool evaluation sessions should take much longer than the pilot studies, perhaps 30 to 60 minutes depending on the number of tasks.

This evaluation plan is subject to change based on feedback from pilot studies and the evolution of the tool’s development. In particular, it may be necessary to perform similar, smaller evaluations for each phase. The other possible change is the evaluation method. The proposed qualitative approach is used due to the lack of easily quantifiable metrics for a successful project search. However, should such metrics arise during pilot studies, the tool evaluation should also make use of them. For example, it is possible that pilot study participants consistently mention that the tool makes selecting projects much faster. In that case, the evaluation may then compare the time to complete tasks between the Google/GitHub search condition and the tool condition. On the other hand, it is entirely possible that pilot study participants consistently mention that using the tool slows them down but reduces the need to click through repository pages. In that case, number of link clicks may be a good metric to compare.

5.5. Tentative Timeline

This proposed work is currently in progress by a team of undergraduate and graduate research assistants. The team is perhaps necessary due to the considerable implementation effort of this project. The following timeline outlines the research plan for the next 12 months.

Oct. 2015	Phase 1: Enhanced Project Summary
	- Minimal Prototype implemented
	- Iterative pilot testing and development
	- Partial evaluation
Jan. 2016	Phase 2: Personalized Project Search
	- Minimal Prototype implemented
	- Iterative pilot testing and development
	- Integration with phase 2
May. 2016	Tool Evaluation
Jun. 2016	Dissertation Writing

REFERENCES

1. A Begel, Yit Phang Khoo, and T Zimmermann. 2010. Codebook: discovering and exploiting relationships in software repositories. *Software Engineering, 2010 ACM/IEEE 32nd International Conference on 1*, 125–134. http://doi.org/10.1145/1806799.1806821

2. danah m. Boyd and Nicole B Ellison. 2007. Social Network Sites: Definition, History, and Scholarship. *Journal of Computer-Mediated Communication* 13, 1: 210–230. http://doi.org/10.1111/j.1083-6101.2007.00393.x

3. Juliet Corbin and Anselm Strauss. 2008. *Basics of qualitative research: Techniques and procedures for developing grounded theory*. Sage.

4. Kevin Crowston, Kangning Wei, James Howison, and Andrea Wiggins. 2008. Free/Libre open-source software development: What we know and what we do not know. *ACM Comput. Surv.* 44, 2: 7:1–7:35. http://doi.org/10.1145/2089125.2089127

5. Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. 2012. Social coding in GitHub: transparency and collaboration in an open software repository. *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, ACM, 1277–1286. http://doi.org/10.1145/2145204.2145396

6. Joan DiMicco, David R Millen, Werner Geyer, Casey Dugan, Beth Brownholtz, and Michael Muller. 2008. Motivations for social networking at work. *Proceedings of the 2008 ACM conference on Computer supported cooperative work*, ACM, 711–720. http://doi.org/10.1145/1460563.1460674

7. Judith Donath. 2005. *Signals, truth, and design*. MIT Press, Cambridge, MA.

8. Judith Donath. 2007. Signals in Social Supernets. *Journal of Computer-Mediated Communication* 13, 1: 231–251. http://doi.org/10.1111/j.1083-6101.2007.00394.x

9. Nicolas Ducheneaut. 2005. Socialization in an Open Source Software Community: A Socio-Technical Analysis. *Computer Supported Cooperative Work (CSCW)* 14, 4: 323–368. http://doi.org/10.1007/s10606-005-9000-1

10. Georgios Gousios, Martin Pinzger, and Arie van Deursen. 2014. An Exploratory Study of the Pull-based Software Development Model. *Proceedings of the 36th International Conference on Software Engineering*, ACM, 345–355. http://doi.org/10.1145/2568225.2568260

11. Carl Gutwin, Reagan Penner, and Kevin Schneider. 2004. Group Awareness in Distributed Software Development. *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work*, ACM, 72–81. http://doi.org/10.1145/1031607.1031621

12. Andrew J Ko and Parmit K Chilana. 2011. Design, Discussion, and Dissent in Open Bug Reports. *Proceedings of the 2011 iConference*, ACM, 106–113. http://doi.org/10.1145/1940761.1940776

13. Georg von Krogh, Sebastian Spaeth, and Karim R Lakhani. 2003. Community, joining, and specialization in open source software innovation: a case study. *Research Policy* 32, 7: 1217–1241. http://doi.org/http://dx.doi.org/10.1016/S0048-7333(03)00050-7

14. Josh Lerner and Jean Tirole. 2002. Some Simple Economics of Open Source. *The Journal of Industrial Economics* 50, 2: 197–234. http://doi.org/10.1111/1467-6451.00174

15. Jennifer Marlow, Laura Dabbish, and Jim Herbsleb. 2013. Impression formation in online peer production: activity traces and personal profiles in github. *Proceedings of the 2013 conference on Computer supported cooperative work*, ACM, 117–128. http://doi.org/10.1145/2441776.2441792

16. Raphael Pham, Leif Singer, Olga Liskin, Fernando Figueira Filho, and Kurt Schneider. 2013. Creating a shared understanding of testing culture on a social coding site. *Proceedings of the 2013 International Conference on Software Engineering*, IEEE Press, 112–121. Retrieved from http://dl.acm.org/citation.cfm?id=2486788.2486804

17. Peter C Rigby, Daniel M German, and Margaret-Anne Storey. 2008. Open source software peer review practices: a case study of the apache server. *Proceedings of the 30th international conference on Software engineering*, 541–550.

18. Peter C Rigby and Margaret-Anne Storey. 2011. Understanding Broadcast Based Peer Review on Open Source Software Projects. *Proceedings of the 33rd International Conference on Software Engineering*, ACM, 541–550. http://doi.org/10.1145/1985793.1985867

19. Anita Sarma, Zahra Noroozi, and Andre van der Hoek. 2003. Palantir: Raising Awareness among Configuration Management Workspaces. *Software Engineering, International Conference on* 0: 444. http://doi.org/http://doi.ieeecomputersociety.org/10.1109/ICSE.2003.1201222

20. Walt Scacchi. 2007. Free/Open Source Software Development: Recent Research Results and Methods.

In *Architectural Issues*, Marvin V Zelkowitz (ed.). Elsevier, 243–295. http://doi.org/http://dx.doi.org/10.1016/S0065-2458(06)69005-0

21. Chris Scaffidi, Chris Bogart, Margaret Burnett, Allen Cypher, Brad Myers, and Mary Shaw. 2010. Using traits of web macro scripts to predict reuse. *Journal of Visual Languages & Computing* 21, 5: 277–291. http://doi.org/http://dx.doi.org/10.1016/j.jvlc.2010.08.003

22. Leif Singer, Fernando Figueira Filho, and Margaret-Anne Storey. 2014. Software Engineering at the Speed of Light: How Developers Stay Current Using Twitter. *Proceedings of the 36th International Conference on Software Engineering*, ACM, 211–221. http://doi.org/10.1145/2568225.2568305

23. Margaret-Anne Storey, Leif Singer, Brendan Cleary, Fernando Figueira Filho, and Alexey Zagalsky. 2014. The (R) Evolution of Social Media in Software Engineering. *Proceedings of the on Future of Software Engineering*, ACM, 100–116. http://doi.org/10.1145/2593882.2593887