Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository

Laura Dabbish, Colleen Stuart, Jason Tsay, Jim Herbsleb
School of Computer Science and Center for the Future of Work, Heinz College
Carnegie Mellon University
5000 Forbes Ave., Pittsburgh, PA 15213
{dabbish, hcstuart, jtsay, jdh}@cs.cmu.edu

ABSTRACT
Social applications on the web let users track and follow the activities of a large number of others regardless of location or affiliation. There is a potential for this transparency to radically improve collaboration and learning in complex knowledge-based activities. Based on a series of in-depth interviews with central and peripheral GitHub users, we examined the value of transparency for large-scale distributed collaborations and communities of practice. We find that people make a surprisingly rich set of social inferences from the networked activity information in GitHub, such as inferring someone else's technical goals and vision when they edit code, or guessing which of several similar projects has the best chance of thriving in the long term. Users combine these inferences into effective strategies for coordinating work, advancing technical skills and managing their reputation.

Author Keywords
Transparency; awareness; coordination; collaboration; open source software development; social computing.

ACM Classification Keywords
H5.3. Information interfaces and presentation (e.g., HCI): Group and organization interfaces.

General Terms
Human Factors; Design.

INTRODUCTION

The internet has become increasingly social in the last ten to fifteen years, but the productivity implications of these changes remain unclear. We can track a person's moment-by-moment status updates on Facebook, location on Foursquare, and updates on Twitter, blogs, and wikis. These applications all have a common set of important functionality. Users can articulate an interest network of people or things by defining a set of individuals or artifacts (like blogs or RSS feeds) to pay attention to. In doing so, users immediately subscribe to a stream of events and actions other individuals take. Thus the social web provides an unprecedented level of transparency in the form of visibility of others' actions on public or shared artifacts. The question remains, however, what benefits this transparency provides, particularly in the large scale (i.e. across a community).

Previous work on awareness has explored the value of activity information for small groups [7, 12]. This work has found that notifying members of actions on shared artifacts helps them maintain mental models of others activities [11] and avoid potential coordination conflicts [20]. However, activity awareness has largely been examined in the context of well-defined small groups within organizations. Online, individuals participate in large-scale, ill-defined communities that often have hundreds if not thousands of members. Transparency of others' actions in this type of setting may have very different benefits as a function of the larger scale and the fact that interactions are no longer embedded in an organizational context.

Visible cues of others' behavior on a social website are likely to support a variety of interpretations about their motivations and the community more generally. People are social creatures and make inferences about others from what they observe (e.g., [27]). Surfacing information about people's actions on artifacts is no longer a technological challenge. What is more interesting, and less understood, is what people are able to infer from such a collection of information, and how these inferences help them carry out their collective work. In this research we were interested in the collaborative utility of activity transparency in a large community engaged in knowledge-based work. We address the following two research questions to advance our understanding of transparency in online social sites:

(1) *What inferences do people make when transparency is integrated into a web-based workspace?*

(2) *What is the value of transparency for collaboration in knowledge-based work?*

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
CSCW'12, February 11–15, 2012, Seattle, Washington.
Copyright 2012 ACM 978-1-4503-1086-4/12/02...$10.00.

In order to address these questions, we examined a successful social site called GitHub (GitHub.com), a code-hosting repository based on the Git version control system. GitHub is an iconic example of a knowledge-based workspace. This site integrates a number of social features that make unique information about users and their activities visible within and across open source software projects. We interviewed light and heavy users of the site, having them walk us through a typical session and describing their activity within projects. We found that they made a rich set of inferences from the visible information, such as inferring someone's technical goals based on actions on code. Developers combined these inferences into effective strategies for coordinating projects, advancing their technical skills and managing their reputation. In the next sections we consider related research, describe the GitHub context and our study, present the results of our interviews, and finally discuss implications of our findings.

THE VALUE OF AWARENESS

Previous work on collaboration and awareness suggests that providing visibility of actions on shared artifacts supports cooperative work in a variety of ways. More recent work has shown the utility of social tools and systems for relationship management in the workplace. However, these literatures have not yet articulated the value of integrating social networking functionality with activity awareness.

Collaboration and awareness

Collaborators who are physically collocated have some level of awareness of each other's activities because of frequent opportunities for interaction [17, 24]. The affordances of collocation, documented through careful field study of software developers [14, 24] and other knowledge-based workers include: overhearing, shared visual space, and shared memory of discussion around artifacts [11, 17, 24]. These affordances support awareness of others' work state and expertise, both useful for coordinated action in collaborative projects [2, 12, 17].

Awareness systems attempt to provide distributed collaborators with the same type of mutual knowledge [11]. These systems have taken various forms with a variety of foci, including informal social awareness to support casual interaction [11], structural awareness of group member roles and status, and workspace awareness of actions on shared artifacts [2, 12, 18]. The theory of social translucence suggests these types of awareness systems are useful because they can make socially significant information visible, support awareness of collaborators' behavior, and make the viewer accountable for that information [8].

For the most part, however, collaborative awareness tools have been designed for and evaluated within small groups. It is unclear to what extent they scale across a wider range. Previous work on awareness and software development has indicated a tradeoff between specific awareness of a very small set of developer actions, versus general awareness across a project as a whole, noting the effort required to proactively provide status information (e.g. [13]).

In addition, collaborative awareness systems have typically utilized momentary notification of the most recent activity as opposed to a history of actions on shared artifacts. Only a handful of systems have attempted to provide visualizations of activity over long periods of time (e.g. history flow system which visualizes collaborator edits to Wikipedia articles [26]). We do not yet understand how these types of activity traces influence collaborative action, particularly in online production settings like Wikipedia where hundreds or thousands of collaborators can be involved on an article.

Social computing and software development

Social computing technologies such as micro-blogging, activity feeds, and social annotations, facilitate lightweight interactive information sharing within the web browser. Social computing technologies shift the focus of interaction to individual contributors and their activities with electronic artifacts. Individuals articulate their interests, likes and dislikes, as well as their social network [4]. When combined with information visualization techniques, these tools may help individuals make sense of activity and contribution on a much wider scale.

Previous work on social media for work purposes has shown that systems like Facebook and LinkedIn are useful for maintaining weak tie relationships and bridging organizational boundaries [6, 21]. And organization internal social networking sites like Beehive in IBM support similar types of boundary spanning activities in an organization [6]. At the same time, these systems have been integrated with work artifacts in a limited way (e.g. bookmarks in [16]).

When social computing technologies are used in a software development context, there is an opportunity to leverage articulated social networks and observed code-related activity simultaneously to support the type of awareness previous only available to collocated teams (e.g. in systems like [9, 18, 19]). Software engineering is only beginning to make exploratory use of social computing technologies to enhance collaboration. For example, tagging [22], searchable graphs of heuristically linked artifacts [3], and workspace awareness [9, 18, 19] have shown some promise for supporting coordination in software development. However, the utility of these technologies has been examined in isolation from the rich ecology of open software development [23]. We know relatively little about how developers actually adopt and adapt these tools in the process of their work, and when and how they improve coordination and performance.

METHOD

Research Setting

In order to address our research questions, we examined collaboration among users of a large, open source software hosting service GitHub [10]. GitHub provides a set of

"social coding" tools built around the Git version control system (http://git-scm.com/) and incorporates social functionality that makes a developer's identity and activities visible to other users. The GitHub site is unique in that it makes user identities, internal project artifacts and actions on them publicly visible across a wide-community.

People

On the GitHub site, developers create profiles that can be optionally populated with identifying information including a gravatar (an image representing them throughout the site), their name, email address, organization, location, and webpage. A developer's profile is visible to other users and displays all the repositories that person is working on and a list of their latest activities on the site (see Figure 1).

Code Artifacts

GitHub currently hosts over one million code repositories, and has 340,000 registered contributors [10]. While a majority of the projects on GitHub are single-developer code dumps, many are active multi-developer projects of significant scale that have been running for some time. Each repository on GitHub has a dedicated project page that hosts the source code files, commit history, open issues, and other data associated with the project. Developers can create permanent URLs to link to specific lines within a code file. This functionality allows information about artifacts within the site to flow outside of the GitHub community to the web at large.

Actions

Actions in GitHub occur when a person changes an artifact or interacts with another person through the site. These actions can be code-related, communication, or subscription. Actions on code or associated with code include committing, forking and submitting a pull request. Project owners can make *commits*, i.e. changes to the code, by directly modifying the contents of code files. Developers without commit-rights to a project must fork a project, creating a personal copy of the code that they can change freely. They can then submit some or all of the changes to the original project by issuing a pull request. The project owner or another member with commit rights can then merge in their changes. Developers can also communicate around code-related actions by submitting a comment on a commit, an issue, or a pull request.

The record of all action information combined with user subscription allows activity updates to flow across the site. Subscription actions include *following* and *watching*. Developers can 'follow' other developers and 'watch' other repositories, subscribing them to a feed of actions and communications from those developers or projects (Figure 2) with frequent updates for active projects.

Actions on artifacts also become artifacts themselves, as the history of user actions on code artifacts is recorded over time. The feed presents a recent history of following, watching, commit, issues, pull requests, and comment actions. Visualizations on the site, such as the network view, provide access to the history of commits over time across all forks of a particular project (see Figure 3).

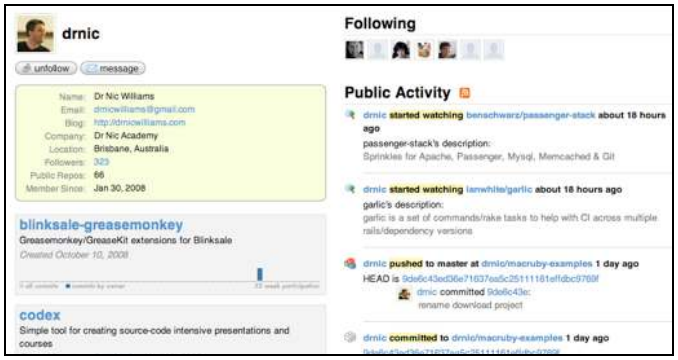

Figure 1. GitHub user profile with projects and public activity

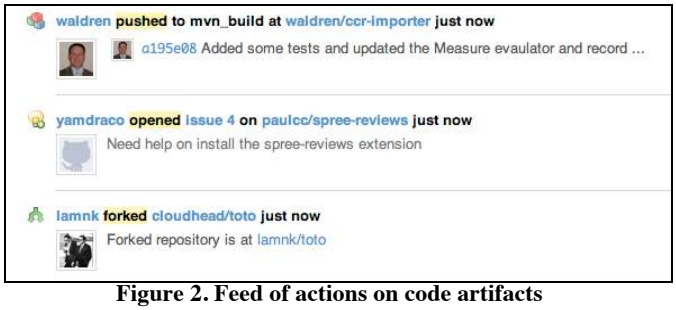

Figure 2. Feed of actions on code artifacts

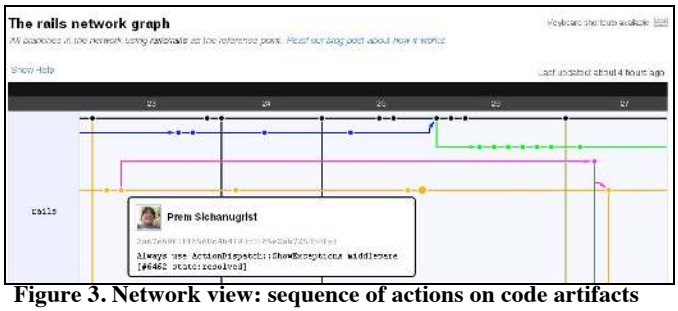

Figure 3. Network view: sequence of actions on code artifacts

To get a sense of how visible information on GitHub influenced the nature of collaboration and interaction, we interviewed a set of developers who use GitHub. We first examined the types of inferences they made based on the visible information in the site, and next examined what types of higher-level activities these inferences supported.

Data Collection

We conducted a series of semi-structured interviews with 24 GitHub users. Our goal in these interviews was to document and understand in more detail the different ways GitHub functionality was used by our participants. We solicited participants via email and conducted our interviews in person or via phone. Participants were chosen for equal representation across peripheral and heavy users (with greater than 80 watchers on at least one of their OSS projects). This was done because we thought that serious and hobby users might have different purposes and strategies, and very different information loads. Table 1 summarizes the 24 participants in our sample.

	Hobbyist	Work use: non-SW org	Work use: SW org
Peripheral users	P2, P7, P10, P18, P23	P6, P12, P16, P24	P5, P17
Heavy users	P11, P15, P21, P22	P3, P4, P9, P14, P19	P1, P8, P13, P20

Table 1. Summary of interview participants.

Participants were asked to walk us through their last session on GitHub, describing how they interpreted information displayed on the site as they managed their projects, and interacted with other users' projects. Remote participants shared their screen during the interview using Adobe Connect so we could ask specific questions about data on the site and users could demonstrate their activities on the site. Interviews lasted approximately 45 minutes to one hour. These interviews were then transcribed verbatim to support further analysis. The interviews, videos and field notes supported our analysis process.

Data analysis

We applied a grounded approach to analyze the transparency related inferences in our interview responses [5]. We first identified instances of these types of inferences in five interview transcripts. For each example analyzed, we identified what information was made visible by the GitHub system, what inferences the participant was making based on that information, and the associated higher-level goal. We then conducted open coding on these responses, comparing each instance with previously examined examples and grouping examples that were conceptually similar. This process revealed categories of transparency related inferences and higher level behaviors these inferences supported. We used this first set of categories to code the remaining interviews, revealing additional categories and refining our original coding scheme to represent the dataset as a whole. We repeatedly discussed the codes and transcripts in a highly collaborative and iterative process. We continued this process until the interviews no longer revealed new behaviors not captured in our existing set of categories (theoretical saturation).

RESULTS

Our analysis revealed that individuals made a rich set of inferences based on information on GitHub. These inferences were a function of four sets of visible cues (summarized in Table 2).

Recency, volume, and location of actions signaling commitment and interests

As with other low-cost hosting sites, GitHub has a mix of projects that are little more than code dumps and serious projects that continue to receive attention and effort. There is also a mix of hobbyists who make occasional contributions and move on, and dedicated developers who provide project stewardship over the longer term. Our interviewees often used the recency and volume of activity as a signal of commitment or investment at the individual and project level.

Visible information about other developers' actions influenced perceptions of their commitment and general interests. Recent activity gave a sense of the level of investment in a project. The feed of developer actions across projects helped other developers infer their current interests. One respondent described following a friend to stay up to date on what he was up to through his commits (P16). The amount of commits to a single project signaled commitment or investment to that project, while the type of commits signaled interest in different aspects of the project.

Visible Cues	Social Inferences	Representative Quote
Recency and volume of activity	Interest and level of commitment	*"this guy on Mongoid is just -- a machine, he just keeps cranking out code." (P23)*
Sequence of actions over time	Intention behind action	"*Commits tell a story. Convey direction you are trying to go with the code … revealing what you want to do*." *(P13)*
Attention to artifacts and people	Importance to community	*"The number of people watching a project or people interested in the project, obviously it's a better project than versus something that has no one else interested in it." (P17)*
Detailed information about an action	Personal relevance and impact	*"If there was something [in the feed] that would preclude a feature that I would want it would give me a chance to add input to it." (P4)*

Table 2. Visible cues and the social inferences they generated.

Recent activity signaling project liveness and maintenance

As with many open source hosting sites, dead and abandoned projects greatly outnumber live ones that people continue to contribute and pay attention to. It can be tedious to figure out which are which, yet it is important to do so, since one does not want to adopt or contribute to a dead or dying project. In GitHub, developers described getting a sense of how 'live' or active a project was by the amount of commit events showing up in their feed.

> *"Commit activity in the feeds shows that the project is alive, that people are still adding code."* (P16)

Users also relied on historical activity to make inferences about how well the project was managed and maintained. Lots of open pull requests indicated that an owner was not particularly conscientious in dealing with people external to the project, since each open pull request indicates an offer of code that is being ignored rather than accepted, rejected or commented upon (P11).

Sequence of actions conveying meaning

Visible actions on artifacts carried meaning, often as a function of their sequence, or ordering with respect to other

actions. When considering the actions of an individual developer, they signaled intentions, competence and experience. In the project context, actions on code and who carried them out allowed others to make inferences about the structure of the project and collaborator roles.

Commits conveying developer intention
At the lowest level, commit information connected with other commits, comments, or issues conveyed meaning or intention behind actions. Several respondents were able to look at a sequence of commits and infer what the developer was trying to accomplish with their changes (e.g. P1, P13, P9, P21, P11). Linking commits and issues similarly communicated the reasoning behind a change to the code.

History of activity signaling competence
At the developer level, information about past commits, number of projects created (versus forked), and activity in those projects all fed into perceptions of developer skill. Several respondents noted that GitHub profiles now act as a portfolio of work and factor into the hiring process at many companies (P3, P9, P13, P8, P17), in part because they provide a sense of a developer's work style and pace.

History of activity signaling project structure and roles
Developers also seemed able to use the history of actions on code to make sense of a project's evolution over time or history. The active public record of who contributed what aspects of the code when, meant developers were able to describe how a project got into its current state, who originally founded the project, what happened across different releases, and who was responsible for what areas of the code. Developers described using this commit history information to infer other's expertise with parts of a project. Respondent P1 indicated that he would look at who made changes to which file to find who had expertise on a piece of the system (essentially inferring who knows what from the record of activities on the project).

Attention signaling community support
Visible cues about who was attending to something served as an important signal of community support (or lack thereof). Developers interpreted activity traces of attention (following, watching, and comment activity) as an indicator that the community cared about that person, project, or action. Signals of attention also seemed to lead to developer attention to that person, artifact or event.

Attention signaling action or artifact importance
Respondents used signals of other users' attention to a feed item as indicators that an artifact or action was important. In particular, comments on a commit suggested the commit was interesting, controversial, or worth looking at (P17). Actions signaling attention to a project or person (watching or following) similarly signaled it or they were interesting in some way and prompted developers to look more closely. Inferences about who would see a particular action also influenced perceived value of engaging in that action. Inference of the size of a potential audience was cited as a motivation to contribute. Respondents indicated the number of forks or watchers of a project signaled that 'lots of other people will benefit from this change' (P13) or that 'someone would find this useful' (P16). The size of the potential audience was also frequently cited as a reason for using GitHub in the first place, based on general community interest in other forums such as mailing lists.

Attention signaling developer status
The number of followers a developer had was interpreted as a signal of status in the community. Developers with lots of followers were treated as local celebrities (e.g. dhh). Their activities were retold almost as local parable; our interviewees knew a great deal about them and paid attention to their actions (P20).

Attention signaling project quality
Visible information about community interest in the form of watcher and fork counts for a project seemed to be an important indicator that a project was high quality and worthwhile. Several respondents indicated using the number of watchers of a project or forks of a project as a signal that a project had community interest, and so was likely to be good or of interest. As one developer put it:

> *"The way you know how useful something is, is how much community there is behind it."* (P23).

These signals of attention, at the same time, put pressure on developers since their visible actions affected attributions about project quality. Several respondents indicated an awareness of being watched, noting that updates about their changes would 'flow to everyone watching' (P13) and that 'everyone can see what you're doing' (P1). In some cases they inferred the identity of this audience (e.g. users of their project) based on who they knew was watching or had forked the project noting for example "people are watching because they depend on it." (P4).

Action details signaling personal relevance
Certain properties of actions in the feed signaled potential personal impact. These inferences were highly dependent on the developer's own work and interests, rather than the community at large.

Actions signaling contribution opportunities
Several respondents inferred contribution opportunities from action information they would see in the feed. For example, one respondent described continually watching the feed for issue submissions or comments on commits (P9), both representing a chance for him to add something to the code or to the ongoing discussion.

Actions signaling potential problems
Respondents also inferred potential problems from commit events they would view in the feed or in the recent commit

list. These inferences were based on cues that a commit was connected with specific files on a project, or had comments suggesting the change would affect their own projects (P16, P19). Comments on a commit also signaled a potentially contentious or problematic change (P1).

SOCIAL INFERENCES INFORMING JOINT ACTION

The social inferences that individual developers made based on visible cues of others' behavior fed into three types of higher-level collaborative activities: project-management, learning through observing, and reputation management.

Project management

All of the developers we interviewed had GitHub projects they were primarily responsible for, either because they were owners of the project or centrally involved in development. Certain types of social inferences described above supported project management activities.

Recruiting developers

Several of the developers we talked with actively recruited others (or were recruited by others) to contribute to a project more heavily. This recruitment was fueled by visible information about the other developers' competence and investment in the project. For example, P8 watched commits occurring in the various forks of his project to identify skilled and committed developers who could contribute more actively. In some cases this was based on past successful contributions to the project and observed commitment. For example P2, a newer member of GitHub was 'recruited' by a project owner after submitting several good commits. The project owner began sending him tasks such as requests to address incoming issues. Intense interest in the project, inferred from a high volume of commit actions in a short period of time, sent a strong signal that a contributor was invested in the project, and could be trusted to contribute more centrally (P17 & P21). In the cases observed this perceived investment seemed to translate into trust, with project owners granting commit rights, allowing new members to influence project vision, and sometimes even turning over ownership to the newcomer after they had demonstrated high levels of investment (P2, P8, P11, P17, P21, P22).

Identifying user needs

Transparency also supported identification of user skills and needs. Here the term user refers to other developers who make use of a particular project in their own work, becoming dependent on that project for certain functionality [23]. Developers inferred user needs by watching their activity in forks (personal copies) of the project (P3, P6, P9, P13, P11). For example, one developer described awareness that users were forking his project to deal with various incompatibilities with a new version of another piece of software they used in concert. He was made aware based on their activity in the forks, which incompatibility issues were particularly problematic for his users (P9).

> *"I saw somebody trying to use it with Rails master I'm like well crap I don't know if it works with Rails master so let me check. So that type of stuff has been useful just to get a sense of the kinds of things people might like to see, you know?" (P9).*

In almost all cases these user modifications represented innovations that extended the project in interesting ways, making it compatible with other systems or more generally useful (P3, P21, P9, P13, P11).

Although observable behavior in the forks provided information about user needs, developers often sought direct interaction with users to get feedback on their needs. Several developers mentioned also posting information about a change (using a direct URL to the code) into a project mailing list (P3, P4, P11). In many cases this was to get user input or buy in for a new design decisions that would go into the next release of a project (e.g. P11). Users would also contact developers directly to let them know about a change they wanted to make to the code or an issue they were having with the project (P1, P3, P9, P11). These interactions helped surface user needs but were also seen as a nuisance in some cases when responses were sought in a private channel such as e-mail rather than a public channel where everyone could observe the interaction.

Managing incoming code contributions

Perhaps the most important project management activity developers engaged in on GitHub was managing incoming code contributions. As noted above, users and other developers could submit changes to a project by forking the project and then making a pull request (a request that the changes be merged back into the master branch). Developers were constantly making decisions about what code to accept back into the project (P1, P2, P4, P8, P9, P11, P16, P19, P21). For very large and popular projects, owners dealt with many pull requests per day (P1, P8, P22). As noted above, they described making inferences about the quality of a code contribution based on its style, efficiency, thoroughness (e.g. were tests included?) and in some cases the submitter's competence (P8, P11, P22). Some developers indicated prioritizing requests that were tied to issues or integrated a feature that users had been requesting (P11, P21). Not all of the projects on GitHub used the pull request mechanism. In some cases because of legacy reasons, patches to a project had to be sent to a mailing list (P14, P24) for approval, where the community would chime in on their acceptability. Interestingly in this way, the GitHub pull request mechanism centralized control over changes by allowing managers to bypass a public mailing list notification and discussion mechanism.

Visibility across forks (or copies) of a project took the pressure off of project owners to accept all changes, and allowed niche versions of a project to co-exist with the official release. Thus contributors could build directly on each other's work, even if the project owner did not approve of the changes. As one developer put it:

"I can ignore bad changes but know that the network of experimenters can continue." (P13)

The cross-fork visibility also meant that project owners could proactively solicit changes from developers as they were working, and use the transparency to track the status of ongoing changes by others. Several respondents indicated using the network view (Figure 3) to identify the leading wave of changes to their project, or the newest code (P9, P11, P21). As noted above, they could see what people were trying to do with their code (P1).

"I would look at this [network] view and actually find folks who had uploaded a patch and say, 'Hey are you planning on sending that back to [my project], this is what I think of it, here's some changes you could make, here's some suggestions,' and that kind of got the ball rolling." (P8)

In some cases, the changes would not be submitted back because the person did not finish doing what they had intended to with the project (P1, P11). Here respondents described pinging the developer to solicit a pull request (P21, or receiving a ping P2), or asking when they would finish the change. In some cases, if the change was novel or useful enough, the project owner would take over the code and finish it themselves (P11).

As with user needs, in many cases, project owners needed to directly communicate around a code contribution. This was sometimes an attempt to solicit and motivate changes as described above. More often, however, this interaction consisted of negotiation around incoming pull requests. There was a clear sense that project owners had a view of the trajectory for the project, and there was a need for others to get buy in before making changes (P1, P9, P16). Project owners would often see potential problems that a code submission would cause with other parts of the code, or with changes they wanted to make in the future. In both cases their reaction was based on implicit knowledge about code organization or their future plans for the code (their 'vision' for the project as they often called it):

"I could tell that was actually going to cause some serious problems down the road, so I just responded. I always thank them because it's a big help when people contribute back, but it wouldn't work so I kind of explained to him why it didn't work." (P19)

This information was not transparent to submitters and could only be elicited through direct communication around the code. Similarly the developer's reasoning behind a change or the organization of a code submission was not always clear to the project owner. In some cases, several rounds of comments around a pull request were required to establish shared understanding of what the developer was trying to accomplish and why (P16). The inline interaction with code supported negotiation around a submission so that in some examples, the developer submitting a change would be able to modify the code he had submitted to address concerns a project owner might have about potential conflicts or conformity to project style norms.

Managing dependencies with other projects

Cross project visibility allowed project owners to proactively manage the dependencies their code had with other projects. Project owners were in almost all cases 'users' of the code of many other project owners, meaning changes to those projects would affect the functioning of their own project (P8, P9, P16, P19). Because of this, they attended closely to change events from projects they were dependent on. They watched for commit events in the feed, and reported paying special attention to new releases (which likely contained new features they could make use of) and changes to files they knew their project used (P8, P9, P16).

"[Popular website] their entire engineering team uses [My Project], and so they keep an eye out for any changes as well, because when I do a release, it breaks something then I essentially broke [Popular website]'s entire development for a day or something." (P19)

In some cases, they were watching for changes they knew were coming because they had heard about them in other forums (mailing lists, blogs etc.), or had discussed them directly with project owners or other developers (P9).

When changes occurred that affected their code, developers often directly contacted the project owner or contributor who had made a specific change (P9) or joined in on discussion about a proposed change (P19). For example, one project owner showed us a case where a third party chimed in on the discussion around a pull request someone else had submitted because the change affected functionality his company depended on (P19).

Developers would also handle conflicting or problematic changes by directly modifying the dependent project to address the problem (P9, P16, P20). Transparency supported this behavior because the code artifacts of the dependent project were open and accessible, and the visibility of changes meant the project owner knew exactly why something was no longer working. The project owners in this case were users of others projects, and then had to lobby and negotiate with the dependent project owner to get their changes accepted (e.g. P9).

Learning from others

Interestingly, the transparency on GitHub supported learning from the actions of other developers. Being able to watch how someone else coded, what others paid attention to, and how they solved problems all supported learning better ways to code and access to superior knowledge.

Following rockstars

Developers in our sample described following the actions of other developers because they deemed them particularly good at coding. They referred to these developers with thousands of followers as 'coding rockstars' (P20) and reported interest in how they coded, what projects they were working on, and what projects they were following. In most cases this was because these developers were deemed

to have special skill and knowledge about the domain (P17, P20) in part as a function of their large following.

Watching watching

Developers also reported interest in which projects other users were looking at, and described certain users as acting almost as curators of the GitHub project space (P1, P4, P18, P16, P22). As one developer put it:

> *"I follow people if they work on interesting projects, [then] I'm interested in the projects they're interested in." (P4)*

Certain developers were deemed to have a knack for finding useful projects in a particular interest area:

> *"This guy has good taste in projects. He curates for me. Watching him is like watching the best of objective C that GitHub has to offer." (P16).*

This interest in finding the 'hottest' new projects through what others were watching highlighted the importance that GitHub users seemed to place on novelty more generally.

> *"I learn about new projects and new technologies way faster than ever before and it's just encouraged me to get dialed in to a bunch of different tech communities I never would have had access to before." (P4)*

Identifying new technical knowledge

Developers were also interested in watching the actions of other developers and other projects to find new technical knowledge. In some cases other projects served as a resource to see how other developers had solved a similar problem to theirs (P17, P23). Developers were also interested in watching development over time within projects that were similar in nature to their own.

> *"When I find a project that solves a problem that I had and I'm going to continue to have then I will watch it" (P19).*

By watching these projects and getting updates on the changes they made as they happened, they learned how their technical 'neighbors' were approaching related problems, informing their own development (P5, P16).

Direct feedback

Developers also learned from others through direct interaction. Through comments on pull requests, developers got feedback about their code from more experienced developers. This was sometimes comments about 'good form' or the 'right' way to do things in terms of coding style or what was normative. This was also feedback about code correctness or more efficient ways of writing the same code (e.g. P1). These interactions helped improve the quality of the code submissions.

Communication also supported learning about another developer's project and getting help with attempts to build on that project (P16). Some developers were extremely forthcoming with this type of help, checking their IRC channels and issue requests constantly to find and address those in need (e.g. P11, P19, P22). For some, this was an opportunity to grow a potential contributor, and project owners saw this as a process of ramping up users to eventually become full-fledged contributors (e.g. P8, P6).

Managing reputation and status

The public visibility of actions on GitHub led to identity management activities centered around gaining greater attention and visibility for work.

Self-promotion

Visibility for work was recognized as a valuable aspect of the GitHub community (P13, P16). The developers we interviewed talked about the positive utility of visibility, which led to increased use of a project, extension by others, ideas from a broader audience and exposure for other projects created by the owner (P16).

At the same time, self-promotion, active attempts to gain additional visibility for work, was recognized as a somewhat distasteful activity and something the developers said they wouldn't do (P17). Regardless, many developers consciously managed their self-image to promote their work through consistent branding (giving their project and blog the same name, or using the same Twitter handle and GitHub user id), and by publicizing their work on other platforms outside of GitHub (P9, P19, P17, P23). As one user noted:

> *"I think a lot of people that use GitHub are trying to promote themselves. This is very self-promotional. It's like I have this project, you will be interested in it" (P16).*

The attention associated with self-promotion was motivating for some of the developers we talked to. One developer noted that watchers kept him working on something he might have otherwise abandoned:

> *"Watching lets me know someone cares" (P17).*

Social capital, identity and recognition

Because watching was recognized as a signal of project quality, it carried meaning as a sign of community approval for a project as well. Several developers we talked to mentioned watching a friend's project to increase their 'social capital' on the site, or promote their work (P23). This was also done explicitly by posting projects to external sites such as HackerNews, a common source of information for developers in the GitHub community, or suggesting a project for RailsCast. Projects featured there were known to receive a boost in watchers. The reciprocal visibility of actions in GitHub meant that a certain amount of face management was associated with behavior on the site. Developers did not want to offend others by, for example publicly rejecting code contributions from long time contributors (P9, P21), or not following someone who followed them (P9, P16).

Being onstage

Many of the heavy users of GitHub expressed a clear awareness of the audience for their actions (P1, P6, P13). This awareness influenced how they behaved and

constructed their actions, for example, making changes less frequently (P6), because they knew that 'everyone is watching' (P1) and could 'see my changes as soon as I make them' (P13). There was a concern to get things right because of how public changes to the code would be. One developer contrasted his heavily watched project with a niche project, noting that he could be more experimental in the niche project because no one was watching (P21). Another developer directly compared it with the pressure of performing:

> *"[3]I try and make sure my commit messages are snappy and my code is clean because I know that a lot of people are watching. ...It's like being on stage, you don't want to mess up, you're giving it your best, you've got your Hollywood smile" (P4).*

DISCUSSION

Three interesting themes cross-cut the observations in our data about the value of visibility and transparency in the GitHub community: the micro-supply chain ecosystem on GitHub, the value of observation versus direct interaction, and the affordances of attention signals.

Visibility across micro-supply chains

We found that transparency in GitHub allowed work to progress and projects to evolve to become more general as a function of micro-supply chain management. Because all artifacts are visible on the GitHub site, users of a particular project can access its contents, and are made aware of changes to the project on a continuous basis. This awareness and visibility supported direct feedback and interaction between project owners and their users, creating what we refer to as a "micro supply chain." Visibility between the supplier (project owner) and consumer (user) meant that the owner could infer more clearly who their user base was, how they were using the project, and when they were having problems. Consumers were notified about changes to the product, meaning they could anticipate problematic modifications and provide immediate feedback about them. Once notified, consumers could directly communicate with the project owner about changes being made to discuss their consequences or request adaptations that would suit their needs. But they could also directly modify the product and customize it to suit their needs with or without direct communication, if they so desired. In contrast to relatively static and sequential supply chain relationships, what emerged was a far more interactive producer-consumer relationship, characterized by reciprocal dependencies [25].

Communication occurs at the limits of transparency

Communication generally seemed to be a response to the limits of transparency, when the information and inferences afforded by transparency were insufficient for the purpose at hand. Users interacted when conflicts arose between two dependent projects, or when negotiating modifications to pull requests. In each case, communication seemed to happen when transparency broke down -- there was certain information developers could not directly observe.

People seemed to work independently until certain events brought them together, making the dependency more salient, such as when a potentially problematic change would show up in the feed, or when a pull request would create problems for other aspects of the code. Direct communication functioned in these cases, much as mutual adjustment, allowing individuals to directly share unobservable information about their rationale (why they were doing what they were doing), and plans (what they were planning to do next), and negotiate mutually compatible solutions to a conflict. These negotiations were supported by direct communication interactions in code comments, IRC channels, campfire, mailing lists, and a variety of other web-based communication tools.

Thus although passive activity traces of others' behavior are powerful in some ways, they are limited when joint action is required. In part this is because of the lack of feedback or interactivity these visible traces provide. Our results suggest these traces support knowing what someone has done, and who might be looking at something, at the individual level, and when new collaborative actions introduce new dependencies, two-way communications are required.

Signals of attention

Visible signals of attention provided notification of other developers' behavior on the GitHub site. Interestingly, these signals of attention seemed to help users manage the downsides of transparency across a large-scale network. Visible cues of what others were watching or commenting helped developers find 'interesting' or 'useful' projects and events (in their words). These signals, when aggregated, also gave some users higher status because they indicated community approval or admiration. As one user put it, "I'm kind of giving them some token of my attention. I'm saying, I like what you're doing" (P23). Signals of attention functioned to provide awareness of what other users cared about or were looking at. This awareness is one aspect of social translucence as described by [8].

These signals of attention also in some cases motivated behavior, giving developers a feeling that someone cared about what they were doing. This connects with the notion of accountability in social translucence and collective effort. This affordance of transparency relates to research investigating how working with others affects one's own productivity through social pressure (e.g., [15]), the flow of ideas [1] and help. Our findings suggest that the visibility of actions might act to facilitate information flows and help-giving, both of which have important implications for the quantity and quality of work.

CONCLUSION

In this work we examined how individuals interpreted and made use of information about others' actions on code in an open social software repository. We found that four key

features of visible feedback drove a rich set of inferences around commitment, work quality, community significance and personal relevance. These inferences supported collaboration, learning, and reputation management in the community. Our results inform the design of social media for large-scale collaboration, and imply a variety of ways that transparency can support innovation, knowledge sharing, and community building.

ACKNOWLEDGEMENTS

This research is supported by the Center for the Future of Work at Carnegie Mellon University's Heinz College and by the National Science Foundation under award: CNS-1040801.

REFERENCES

1. Azouley, P., Graff Zivin, J.S., & Sampat, B.N. The diffusion of scientific knowledge across time and space. *NBER Working Paper Series* (2011).
2. Bardram, J. E. & Hansen, T. R. Context-based workplace awareness. *Computer Supported Cooperative Work (CSCW) 19*, 2 (2010), 105-138.
3. Begel, A., DeLine, R., & Zimmerman, T. Social media for software engineering. In *Proc. FoSER 2010*, IEEE Computer Society (2010), 33-38.
4. boyd, d.m., & Ellison, N.B. Social network sites: Definition, history, and scholarship. *Journal of Computer-Mediated Communication*, *13*, 1 (2007), article 11.
5. Corbin, J.M., & Strauss, A.L. Basics of qualitative research. London, UK: Sage Publications, 2008.
6. DiMicco, J., Millen, D., Geyer, W., Dugan, C., Brownholtz, B. & Muller, M. Motivations for social networking at work. In *Proc CSCW 2008,* ACM Press (2008), 711-720.
7. Dourish, P. & Bellotti, V. Awareness and coordination in shared workspaces. In *Proc. CSCW 1992*, ACM Press (1992), 107-114.
8. Erickson, T., & Kellogg, W. Social translucence: An approach to designing systems that support social processes. *TOCHI 7*, 1 (1999), 59-83.
9. Froehlich, J., & Dourish, P. Unifying artifacts and activities in a visual tool for distributed software development teams. In *Proc. ICSE 2004*, IEEE Society (2004), 387-396.
10. Github: Social Coding, http://github.com/, accessed March 17, 2011.
11. Gross, T., Stary, C., & Totte, A. User-centered awareness in computer-supported cooperative work systems. *Int. J. of HCI, 18*, 3 (2005), 323-360.
12. Gutwin, C., Greenberg, S., & Roseman, M. Workspace awareness in real-time distributed groupware: Framework, widgets, and evaluation. In *Proc. HCI on People & Computers,* Springer-Verlag (1996), 281-298.
13. Gutwin, C., Penner, R., & Schneider, K. Group awareness in distributed software development. In *Proc CSCW 2004*, ACM Press (2004), 72-81.
14. Ko, A.J., DeLine, R., & Venolia, G. Information needs in collocated software development teams. In *Proc ICSE 2007*, IEEE Computer Society (2007), 344-353.
15. Mas, A., & Moretti, E. Peers at work. *American Economic Review, 99*, 1 (2009), 112-145.
16. Millen, D., Feinberg, J., & Kerr, B. Dogear: Social bookmarking in the enterprise. In *Proc. CHI 2006*, ACM Press (2006), 111-120.
17. Olson, G., M., & Olson, J.S. Distance matters. *Human-Computer Interaction 15* (2001), 139-178.
18. Omoronyia, I., Ferguson, J., Roper, M. & Wood, M. Using developer activity data to enhance awareness during collaborative software development. *Computer Supported Cooperative Work (CSCW) 18*, 5 (2009), 509-558.
19. Sarma, A., Maccherone, L., Wagstrom, P., & Herbsleb, J. (2009). Tesseract: Interactive visual exploration of socio-technical relationships in software development. In *Proc. ICSE 2009*, IEEE Computer Society (2009), 23-33.
20. Sarma, A., Z. Noroozi, & Htreetreeoek, A. (2003). Palantír: raising awareness among configuration management workspaces. In *Proc ICSE 2003*, IEEE Computer Society (2003), 444-454.
21. Skeels, M.M., & Grudin, J. When social networks cross boundaries: A case study of workplace use of Facebook and LinkedIn. In *Proc. GROUP 2009*, ACM Press (2009), 95-103.
22. Storey, M.A., Ryall, J., Singer, J., Myers, D., Cheng, L.T., & Muller, M. How software developers use tagging to support reminding and refinding. *IEEE TSE 35*, 4 (2009), 470-483.
23. Storey, M., Treude, C., van Deursen, A., & Cheng, L.T. The impact of social media on software engineering practices and tools. In *Proc. FoSER 2010*, ACM Press (2010), 359-363.
24. Teasley, S. Covi, L. Krishnan, M. & Olson, J. Rapid software development through team collocation. *IEEE TSE 28*, 7 (2002), 671–683.
25. van de ven, A.H., Delbecq, A.L., & Koenig, R. Determinants of coordination modes within organizations. *American Sociological Review*, (1976).
26. Viegas, F., Wattenberg, M., & Kushal, D. Studying cooperation and conflict between authors with *history flow* visualizations. In *Proc CHI 2004*, ACM Press (2004), 575-582.
27. Weiner, B. *Human Motivation: Metaphors, Theories and Research*, Newbury Park, CA: Sage Publications, 1992.