Implicit Coordination: A Case Study of the Rails OSS Project

Kelly Blincoe(✉) and Daniela Damian

University of Victoria, Victoria, Canada
kblincoe@acm.org, danielad@uvic.ca

Abstract. Previous studies on coordination in OSS projects have studied explicit communication. Research has theorized on the existence of coordination without direct communication or *implicit coordination* in OSS projects, suggesting that it contributes to their success. However, due to the intangible nature of implicit coordination, no studies have confirmed these theories. We describe how implicit coordination can now be measured in modern collaborative development environments. Through a case study of a popular OSS GitHub-hosted project, we report on how and why features that support implicit coordination are used.

1 Introduction

There are many large Open Source Software (OSS) development projects that succeed despite spanning geographic, organizational and social boundaries. Such boundaries normally create coordination barriers and make coordination more expensive [1]. Previous research hinted at the promise of implicit coordination, defined as coordination *"reached without discursive communication, shared plans or even previous commitment among the actors"* [2], in reducing coordination overhead on OSS projects [2–4]. Bolici et al. [2] identified cases where dependencies between tasks existed with no evidence of explicit communication and theorized that implicit coordination was used to fulfill these dependencies. However, since implicit coordination occurs without explicit communication, it is difficult to examine, and no studies have confirmed these theories.

Modern software development tools provide unprecedented support for implicit coordination in OSS projects. Developers can understand relevant tasks without interrupting the developers assigned to those tasks for explicit communication. For example, details of a dependent task can be reviewed to gain awareness about the task. Developers can document all task related decisions within task reports and insert comments directly into the source code, all of which can be easily reviewed by others. Additionally, tools like GitHub provide notifications of project activity to help developers stay aware. This transparency supports implicit coordination and makes collaboration easier [5].

We describe how modern development environments enable implicit coordination. We report on a case study of a popular OSS project hosted on GitHub. GitHub is a code hosting service and collaboration environment. Its transparency

E. Damiani et al. (Eds.): OSS 2015, IFIP AICT 451, pp. 35–44, 2015.
DOI: 10.1007/978-3-319-17837-0_4

[5] and built-in social features enable implicit coordination. In a mixed-method research approach, we surveyed 986 developers, interviewed 14 developers and conducted a repository analysis. We find that two GitHub features that support implicit coordination —issue subscriptions and following relationships —are used frequently and report on how and why these features are used.

2 Implicit Coordination

Previous studies on coordination in OSS projects [6,7] studied explicit coordination mechanisms likes emails and bug report comments. Coordination that occurs without explicit communication is known as implicit coordination, which consists of consequential communication and feedthrough [8]. Consequential communication is watching a developer complete their tasks to learn about their activities. Feedthrough is obtaining information about tasks by examining changes to artifacts and is an example of stigmergy. Stigmergy is a concept from biology that states *"work done by one agent provides a stimulus that entices other agents to continue the job"* [4]. Stigmergy occurs when enough information is contained within an artifact or a task report to enable a new developer to complete an ongoing task or start a new dependent task without explicit coordination. This occurs frequently on OSS projects [4]. Even independent tasks build on the work of others [9], so stigmergy plays a key role in OSS development.

Awareness is *"an understanding of the activities of others, which provides a context for your own activity"* [10]. Developers need to be aware of tasks (and associated artifact changes) and people [11]. A lack of awareness can result in coordination breakdowns [12,13]. Previous research found that, on OSS projects, developers obtain awareness through explicit communication [8], but it can also be achieved through implicit coordination. Modern software development tools make development work more visible and transparent [5] providing awareness and potential for implicit coordination. Therefore, the study we present here investigated awareness and implicit coordination and was guided our research question:

How are the features that enable implicit coordination being used on modern software development environments?

We describe features of modern software development environments that support awareness of tasks and people and enable implicit coordination in OSS projects. We then describe our study to address this research question.

3 Implicit Coordination Enabled by Modern Development Environments

While explicit coordination is characterized by communication, implicit coordination is achieved by obtaining awareness of the information needed to complete a task without communication. Modern software development environments have introduced features that enable implicit coordination. We describe how modern

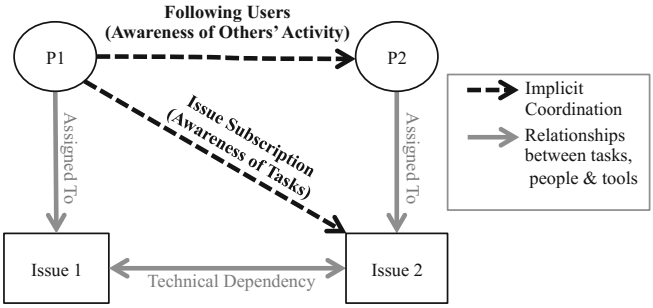

Fig. 1. Awareness Mechanisms for Implicit Coordination in GitHub

development environments support implicit coordination by enabling awareness of tasks and people. We specifically highlight the GitHub features that support implicit coordination and illustrate them in Fig. 1. Issues are GitHub's representation of tasks, so we refer to issues and tasks synonymously. In Fig. 1, a coordination need exists between developers P1 and P2 due to the technical dependency that exists between their tasks.

1) Awareness of Tasks

In a previous study, we found that developers prefer to review task details to understand a task rather than interrupting the task assignee to ask about the task [14]. Reviewing details of related tasks and the changes made to the source code as a result of those tasks can help developers gain an understanding of dependent tasks. Task details can be obtained from the task report in the team's issue or bug tracker. Reviewing artifacts in the source code management system can make developers aware of code changes, and comments left in the code may provide insight into why the changes were made.

Developers can obtain awareness of tasks by ***subscribing to feeds***. Feeds broadcast project-related or user events such as incoming issues or code changes. Developers use feeds to track work and get information [15].

Support in GitHub: GitHub supports awareness of tasks through its issue subscription feature. When users are subscribed to issues, they receive notifications of the activity occurring around that issue on their GitHub dashboard and, if configured, via email. Users are automatically subscribed to issues when they comment on or are tagged in a comment on that issue. Users can unsubscribe if desired. In Fig. 1, developer P1 becomes aware of task 2 through issue subscription.

Other Development Environments/Tools: SourceForge, another web-based code management tool, allows users to subscribe to issues through an RSS feed. Other issue trackers, like Jira and Bugzilla also allow users to obtain notifications of the activity occurring around issues. Jira has an issue subscription feature similar to the one provided in GitHub. The cc feature in Bugzilla works similarly, allowing developers to add themselves to a change request's cc list to receive notifications about that change request.

2) Awareness of Others' Activity

Developers can gain an understanding of others' activity by ***reviewing their work***. This was previously accomplished by monitoring version control check-in logs [8] and has been made easier through feeds that broadcast user activity. Many software development tools allow users to *follow* users. When someone follows a user, they receive notifications about that user's activity in their feeds.

Support in GitHub: Users can follow others by clicking on the follow button in a user's profile. The follower will then receive notifications about that user's activity across all GitHub projects. In Fig. 1, developer P1 is following developer P2 and, therefore, is aware of the activity of P2.

Other Development Environments/Tools: SourceForge enables following of other users through RSS feeds. In Bugzilla and Jira, you can search for issues or bugs a user is participating on, but feeds of a user's activity is not available.

Developers can gain a further understanding of others' activity by ***reviewing information shared through social media***. Software developers have adopted social media tools like *wikis, blogs* and *microblogs*. Studies have found that sharing information through these forums allows easy access to knowledge and can serve as a coordination method [16].

Support in GitHub: GitHub does not allow developers to share information about their activity in wikis, blogs or microblogs. External tools are often used in conjunction with GitHub such as Twitter [16] for this functionality.

Other Development Environments/Tools: No currently adopted development tools offer integrated blogging or microblogging.

4 Case Study

We examined Ruby on Rails, a popular project hosted on GitHub, often referred to simply as Rails. Rails is an open source web application framework written in the ruby programming language. Many companies and other OSS projects use the framework for creating web applications. Rails, therefore, attracts many contributors looking to fix bugs affecting their own products or add new features that are useful for them. We choose to study Rails since it has a large group of code contributors, resulting in frequent coordination needs.

To answer our research question, we collected and analyzed both qualitative and quantitative data through a survey, developer interviews, and a statistical analysis of Rails project repository data. We applied a grounded theory approach on the survey and interview data [17]. For the project repository analysis on Rails, we obtained data from GHTorrent [18], which provides a mirror of the GitHub API data. GHTorrent obtains its data by monitoring and recording GitHub events as they occur. We selected the project with the most code contributors in the GHTorrent 2014-04-02 dataset —Rails. We analyzed the most recent release available in that dataset, 4.0, which was developed from January 20, 2012 to June 25, 2013. Using GHTorrent, we obtained issue subscription information for all issues and following relationships for all users. We included

data from the main branch and any associated forks as recommended in [19]. We included all 2,437 issues that were closed during that release. There were 7,935 users who contributed through commits, comments, or issues. We refer to users and contributors interchangeably since we are studying only Rails contributors.

4.1 Methods

To answer our research question, we analyzed data from an online survey, interviews with 14 contributors, and the project repository.

Survey instrument. We sent an online survey to all 7,492 GitHub users with valid email addresses that participated on Rails during the release of interest. Any user who performed one of the following actions was identified as a participant: committed code; created an issue; submitted a pull request; commented on a commit, issue or pull request; closed, merged or reopened an issue or pull request; or subscribed to an issue.

We asked survey participants if, how and why they used the features that support implicit coordination in GitHub, subscribing to issues and following users. The questions in the survey were both multiple-choice and open-ended. Our survey is available at http://web.uvic.ca/~kblincoe/survey.pdf. We used unbalanced (skewed towards the positive) rating scale questions since we expected mostly positive answers and wanted to measure the degree of the responses [20]. We received 986 responses (12.4% response rate). We used standard qualitative coding techniques [17] to categorize responses and identify themes.

Interview instrument. To gain a better understanding of how implicit coordination takes place, we interviewed 14 of the survey respondents who volunteered for interviews. We randomly selected participants from the 19.2% of respondents who identified themselves as currently active participants of the Rails project. Interviews were semi-structured and lasted 30 minutes on average. The interviews were focused on coordination with questions like 'How do you know what others are working on?' and 'Are there ways you stay aware of project activity and avoid duplicate work or conflicts in your own work without explicitly communicating with other teammates?' Similar to the survey data, we used a grounded theory approach in our analysis of the interview transcripts [17].

Repository analysis. We examined how the features that support implicit coordination are used in GitHub by examining their frequency of use.

4.2 Results

1) Awareness of Tasks: Issue Subscription

Issue subscriptions are used extensively on Rails. 79.7% of issues have at least one subscriber. Of the issues with subscriptions, many (35.6%) have subscribers who have not commented or been tagged within a comment. For issues with subscribers, there is an average of 4.1 subscribers per issue (median is 3).

Many contributors subscribe to issues. Through repository analysis, we found that 48.1% of contributors are subscribed to at least one issue. Users who subscribed to issues were subscribed to 3.9 project issues on average (median is 1).

Our interviewees talked about how the notifications make it easier to stay aware of what is going on. While you can see what others are working on by looking through the code to identify changes, issue notifications are much more efficient.

"Trying to look at the actual code, like the todos or comments in the code or that people have open branches on that, that ends up being really difficult. You can spend a long time looking and not figure it out." [P10]

Many contributors subscribe to issues they are not participating on. Nearly half (49.4%) of our survey respondents said that they subscribe to issues that they are not actively participating on (where active participation considers reporting the issue, developing code or commenting on the issue). This is important because it implies that they are using these notifications for awareness of tasks that they are not working on (like developer P1 in Fig. 1 becomes aware of Issue 2 by subscribing to that issue even though she is not working on that issue).

"[Issue notifications] are the best way to passively stay up to date with things that are going on." [P11]

In fact, many of our survey respondents (65.9%) stated that they use the information obtained from issue notifications differently than when they are actively participating on an issue. When actively participating on an issue, they "use the notifications for more direct actions rather than just a general feeling of what's going on." They treat the notifications as a 'to-do' list. When they are not actively participating on the issues, they use the notifications more passively. They gave several reasons why they would subscribe to notifications for issues they are not participating on:

- *Dependencies.* 31% of respondents explicitly state that they subscribe to issues because of dependencies in their code or because they are experiencing the same issue. One respondent stated, "Something I'm working on could be depending on the outcome of the resolution." Another respondent said, "I ran into the same issue ... when they fix it, I will update to [the new version]."
- *Issue Status.* 31% of respondents simply reported that they subscribe to know when the issue is resolved without giving a more detailed reason.
- *Project Status.* 22% subscribe to maintain an overall awareness of project activity. "For some projects that I'm not actually interested in contributing actively, [subscribing to issues] is the closest you can get to a newsletter to get to know what new features are getting in and other general stuff."
- *General Interest / Education.* 9% subscribe because of general interest in the issue or to learn from the discussion or solution. One respondent said, "I like to see the changes and comments made by others to learn more."
- *Awareness for Future.* 7% subscribe to obtain knowledge for future development. One respondent said the notifications around certain issues "may help me make changes in the area in the future more easily". Other respondents noted they follow certain issues so they can offer help if needed.

One of the most common reasons for receiving notifications (31%) is to obtain information on dependencies. This is a form of implicit coordination. The respondents who receive notifications to provide awareness for the future (7%) are also

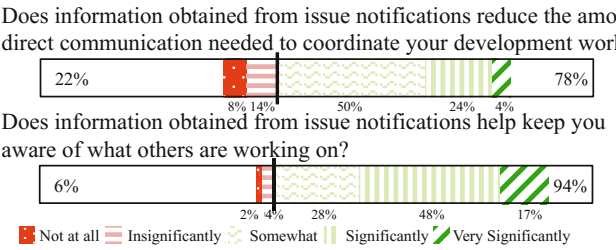

Fig. 2. Survey Responses around Issue Notifications

implicitly coordinating. They obtain awareness about code changes so they are familiar with the code and the design decisions to be able to contribute in the future more efficiently. Only a small number of our survey respondents subscribe to issues out of general interest (9%). Our interviewees only subscribe to issues that affect them directly.

> *"I used to try to follow things that I was just interested in to know what was going on with them, and I found that it was totally irrelevant. If I'm not actively using a repository, I really don't need the notifications."* [P5]

Subscribing to issues helps reduce communication and increase awareness. Fig. 2 shows that 78% of survey respondents found that the information obtained from issue notifications can reduce the amount of direct communication and 94% said it can increase awareness of what others are working on. One respondent said, "I can stay up-to-date on others thinking and inputs on a specific topic without needing to talk to them." Another respondent focused on the awareness he obtains from the information contained in issue notifications saying, "through their comments and patches I can see what others are working on and what their progress is." Our interviewees talked about how the notifications help reduce direct communication around issue status.

> *"If I am waiting for a fix on something, then the notifications will help me stay up-to-date . . . I don't have to contact the developer for the status."* [P5]

Interviewees also noted that communication is often limited to only when differences of opinion occur.

> *"I might see that someone is working on a particular issue and . . . take a look at the code that has actually been implemented and see if it is along the same lines as what I was thinking, kind of a preview. So if somebody is off on the wrong track or going down a code path that I don't think is actually going to fix the bug or implement that feature then I might send them an email or hop in a chat."* [P9]

The notifications also help keep developers aware of dependencies or work that has been done on related issues, according to our interviewees.

> *"If we have multiple pull requests that rely on each other. . . just seeing comments come across and the pull requests come in, helps me know where that is in the process."* [P6]

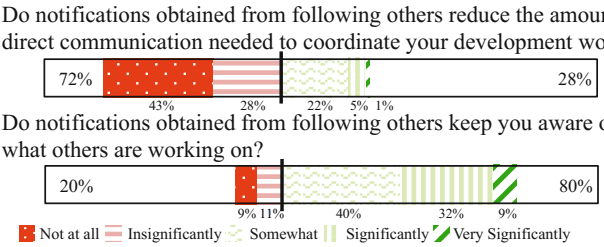

Fig. 3. Survey Responses around Following

2) Awareness of Others' Activity: Following

Many users follow other users. 66.7% of survey respondents say they follow other GitHub users. This is consistent with our repository analysis, which found that 64.3% of all Rails contributors follow other GitHub users. The reasons for following others are quite varied:

- *Track activity.* 41.8% follow others to see what they are up to and track their activity. "As a means of seeing what they are working on. Their contributions are working towards solutions in my general problem space." This reason is the most suggestive of implicit coordination, unlike the ones that follow.
- *Learn about new projects.* 23.8% follow others to see what projects they contribute to or star. "I have discovered a lot of projects from looking at what users I am following are looking at."
- *Social.* Many respondents (21.2%) said they follow others for social reasons. "That's basically the 'friend' feature on GitHub. I just follow others I know."
- *Useless.* Some survey respondents (6.8%) stated that the following feature is useless. "[It] is not really a useful feature because it adds too much noise."
- *Education.* 3.6% said they follow others to learn. "I follow experts in certain code bases to educate myself by reviewing their check-ins and comments."
- *Bookmark users.* 2.8% said they follow others to 'bookmark' interesting GitHub users since it "makes it easier to find their profiles in future."

Users often follow other contributors on their projects. 46.1% of Rails contributors are following other Rails contributors. Those who are following other contributors are following 7.2 project contributors on average (median is 3).

Following other users does not reduce communication but does increase awareness. Fig. 3 shows 72% of survey respondents found that the notifications obtained from following others does not reduce the amount of direct communication. However, 80% found the notifications did increase awareness of what others are working on. One respondent said, "I can see other people's activity and what projects they are actively working on." Our interviewees noted that the notifications from following others is too high-level to be useful for coordination.

> *"I just see the repositories they create, the repositories they fork, what they star and stuff like that. . . . It is just a general overview of what they are doing, and if I want to know anything in detail, I have to still ask them or go to them or just look at the code to see what's going on."* [P2]

In addition, since following others results in notifications related to their activity on all repositories, the notifications can contain a lot of noise.

"When I follow a person, they work on an assortment of repositories, most of which have nothing to do with me probably. I follow very few people." [P5]

5 Discussion

The findings from our study of Rails, a prominent OSS project hosted in GitHub, indicate that both issue subscription and following other users are widely adopted by OSS users. There are various reasons why users choose to use these features. The main reason for subscribing to issues is to obtain information on dependencies, a form of implicit coordination. Survey respondents believed that subscribing to issues reduced their direct communication.

However, many of our interviewees and survey respondents indicated that notifications from following others introduced too much noise and, therefore, were not useful. Additional surveys or interviews could shed some light on what notifications are most useful for implicit coordination so the notifications can be minimized to only the most relevant information. Important research questions include: What is the most useful information for inclusion in notifications of other's activity? Are users influenced by the actions of the users they follow?

The effect of explicit coordination has been studied by quantitatively assessing how the coordination structure of a team aligns with the teams' coordination needs. Conway's Law [21] was the first to introduce the idea of such an alignment. Now that implicit coordination can be measured through features like issue subscription and following, future research can study the impact of implicit coordination. If implicit coordination improves productivity and quality, managers can encourage implicit coordination to reduce coordination overhead. Further, tools that provide coordination recommendations to developers could focus on less expensive, implicit means of coordination. While bringing new knowledge about indirect collaboration in modern, open development environments, this study is only the beginning of our exploration into indirect coordination. We investigated only one project in detail; thus, it suffers from some threats to validity regarding generalizability. Our survey respondents and interviewees were all Rails contributors and were self-selected. We reached saturation in our results, but they may not generalize to other GitHub projects. However, many of our interviewees were active contributors to many GitHub projects and their responses drew on their experience across multiple projects. Additional studies can continue this investigation on other GitHub projects. Further, this investigation can be continued through future studies of implicit coordination in other modern development environments like SourceForge, Jazz or Bitbucket.

References

1. Herbsleb, J.D.: Global software engineering: The future of socio-technical coordination. In: FSE 2007, pp. 188–198. IEEE Computer Society (2007)
2. Bolici, F., Howison, J., Crowston, K.: Coordination without discussion? socio-technical congruence and stigmergy in free and open source software projects. In: STC 2009 (2009)
3. Elliot, M.: Stigmergic collaboration: The evolution of group work. m/c journal **9** (2006)
4. Heylighen, F.: Why is open access development so successful? stigmergic organization and the economics of information. arXiv preprint cs/0612071 (2006)
5. Dabbish, L., Stuart, C., Tsay, J., Herbsleb, J.: Social coding in github: transparency and collaboration in an open software repository. In: CSCW 2012, pp. 1277–1286. ACM (2012)
6. Bird, C.: Sociotechnical coordination and collaboration in open source software. In: ICSM 2011, pp. 568–573. IEEE (2011)
7. Crowston, K., Wei, K., Li, Q., Eseryel, U.Y., Howison, J.: Coordination of free/libre and open source software development (2005)
8. Gutwin, C., Penner, R., Schneider, K.: Group awareness in distributed software development. In: CSCW 2004, pp. 72–81. ACM (2004)
9. Howison, J., Crowston, K.: Collaboration through open superposition: A theory of the open source way. MIS Quarterly **38**, 29–50 (2014)
10. Dourish, P., Bellotti, V.: Awareness and coordination in shared workspaces. In: CSCW 1992, pp. 107–114. ACM (1992)
11. Ko, A.J., DeLine, R., Venolia, G.: Information needs in collocated software development teams. In: ICSE 2007, pp. 344–353. IEEE CS (2007)
12. Damian, D., Izquierdo, L., Singer, J., Kwan, I.: Awareness in the wild: Why communication breakdowns occur. In: ICGSE 2007, pp. 81–90. IEEE (2007)
13. de Souza, C.R., Redmiles, D.F.: An empirical study of software developers' management of dependencies and changes. In: ICSE 2008, pp. 241–250. ACM (2008)
14. Blincoe, K., Valetto, G., Damian, D.: Facilitating coordination between software developers: A timely and efficient approach. Technical Report DCS-354-IR (2014)
15. Treude, C., Storey, M.: Awareness 2.0: staying aware of projects, developers and tasks using dashboards and feeds. In: ICSE 2010, pp. 365–374. IEEE (2010)
16. Singer, L., Figueira Filho, F.M., Storey, M.A.D.: Software engineering at the speed of light: how developers stay current using twitter. In: ICSE 2014, pp. 211–221 (2014)
17. Corbin, J., Strauss, A.: Basics of qualitative research: Techniques and procedures for developing grounded theory. Sage (2008)
18. Gousios, G., Spinellis, D.: Ghtorrent: Github's data from a firehose. In: MSR 2013, pp. 12–21. IEEE (2012)
19. Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D.M., Damian, D.: The promises and perils of mining github. In: MSR 2014, pp. 92–101. ACM (2014)
20. Parasuraman, A., Grewal, D., Krishnan, R.: Marketing research. Cengage Learning (2006)
21. Conway, M.E.: How do committees invent. Datamation **14**, 28–31 (1968)