# Open Source-Style Collaborative Development Practices in Commercial Projects Using GitHub

Eirini Kalliamvakou, Daniela Damian, Kelly Blincoe, Leif Singer and Daniel M. German
Department of Computer Science, University of Victoria
Email: ikaliam@uvic.ca, danielad@uvic.ca, kblincoe@acm.org, lsinger@uvic.ca, dmg@uvic.ca

***Abstract*—Researchers are currently drawn to study projects hosted on GitHub due to its popularity, ease of obtaining data, and its distinctive built-in social features. GitHub has been found to create a transparent development environment, which together with a pull request-based workflow, provides a lightweight mechanism for committing, reviewing and managing code changes. These features impact how GitHub is used and the benefits it provides to teams' development and collaboration. While most of the evidence we have is from GitHub's use in open source software (OSS) projects, GitHub is also used in an increasing number of commercial projects. It is unknown how GitHub supports these projects given that GitHub's workflow model does not intuitively fit the commercial development way of working. In this paper, we report findings from an online survey and interviews with GitHub users on how GitHub is used for collaboration in commercial projects. We found that many commercial projects adopted practices that are more typical of OSS projects including reduced communication, more independent work, and self-organization. We discuss how GitHub's transparency and popular workflow can promote open collaboration, allowing organizations to increase code reuse and promote knowledge sharing across their teams.**

## I. Introduction

GitHub is a popular online code hosting service built on top of git, a decentralized version control system (DVCS). It provides an open development environment and visibility of project activity through both notifications and a simple interface. This transparency promotes increased awareness and reduced communication [12]. Therefore, GitHub has the potential to mitigate challenges distributed projects face in collaboration such as coordination and communication breakdowns [6], [7], [19], a lack of awareness [34], [35], [38], and code conflicts [33]. In fact, GitHub's motto is "collaboration without upfront coordination" [1].

The fact that GitHub claims to mitigate coordination hurdles in today's largely distributed teams is intriguing. Through a surge of researcher interest in recent years, we have learnt about the use of GitHub as a hub for software development activity, such as its use of social and communication features [12], [25], [41], [43] and the impact of its technical features on development practices [16], [30]. These insights are, however, mostly from Open Source Software (OSS) projects hosted on GitHub. GitHub's workflow and development model seem to fit the OSS-like practices which are aligned with GitHub's motto of minimized coordination. Such practices include self-organization, independent work and lightweight communication [18]. At the same time, a fast-growing number of commercial projects also use GitHub as a development environment including large corporations such as Lockheed Martin, Microsoft, LivingSocial, VMware, and Walmart [4], [21], [26]. Most commercial organizations do not allow public access to their code base, so they use private repositories or deploy GitHub's enterprise version on their own servers. We are not aware of any research studies that have investigated the support GitHub provides to these projects.

Understanding collaboration practices in teams using GitHub to produce proprietary software in commercial organizations is equally interesting and important because traditional practices of closed source projects do not seem tailored for the GitHub development environment which promotes visibility of work and self-assignment of tasks. Traditionally, developer participation in closed source projects is by invitation only - developers are assigned tasks and, in extreme cases, are only authorized to work on the code associated with their tasks. On these projects, developers are also typically not aware of code changes made outside their assigned code areas [44]. In this paper we report from a study of collaboration in GitHub in which we analyze *how teams building proprietary software in commercial organizations use GitHub's features and how collaboration is impacted by their use*.

Our study consisted of an online survey of 240 developers and semi-structured interviews with 30 GitHub users; 24 of them were developers, project managers or CTOs that use GitHub in commercial projects that build proprietary software. We found that many commercial software projects are successfully adopting practices like independent work, reduced communication and coordination requirements and self-organization which are common in OSS projects [12], [17], [23], [31]. Even though multiple workflows are supported in GitHub, the teams that our study participants worked in converged towards essentially the same workflow in which developers do not make commits directly to the master code base, and instead commit them to branches or forks. While this type of workflow is known to be in use in OSS projects [16], it is surprising to see it is also adopted by commercial projects since its original purpose was for gatekeeping when there is a lack of trust or need for screening incoming contributions. We discuss how this workflow and adopting OSS-style practices affects five elements of collaboration: coordination, task division, awareness, communication, and conflict resolution.

We further discuss how GitHub's characteristics can support open collaboration [30] inside organizations. Companies see benefit in dropping the barriers between multiple teams and projects that they run internally, promoting inter-team collaboration. GitHub can support open collaboration within commercial organizations by centralizing tools and information and making them transparent. Advocated by developers acting as change agents, the organic adoption of the tool and its

[1] https://help.github.com/articles/using-pull-requests/

corresponding process can ensure that collaborators have a common toolkit to work with.

## II. Background & related work

Collaborative software development projects bring together the interdependent efforts of team members, coordinated towards a common goal [45]. Malone and Cowston [24] view collaboration as building on efficient coordination of direct or indirect actions needed to manage interdependencies.

Collaboration brings known challenges, along a number of dimensions. *Coordination* and *communication* often break down in large and distributed teams, and result in build failures and longer resolution times [6], [7], [19]. Maintaining *awareness* of interdependencies is also challenging; tools can provide alerts of potential coordination needs and recommend communication between interdependent developers [34], [35], [38], but it is not without overhead [5]. *Conflicts* occur even in the presence of awareness tools and teams spend effort to resolve them [33]. Efficient *task division* and scheduling can help to minimize, but not eliminate, coordination overhead and code conflicts [20]. Modularization of tasks is an accepted way to minimize interdependencies [46], but it is impossible to remove them altogether.

These collaboration challenges are typical in software development, and they are accentuated when teams are distributed. Collaborative Development Environments (CDEs) integrate source code management tools and bug trackers with collaborative development features [22] and have been suggested as a solution to the communication and coordination challenges of distributed software projects in organizations [1]. GitHub is an example of a CDE that provides an infrastructure for collaborative development. Besides its code hosting service, it offers collaborative code review mechanisms, an integrated issue tracker and social features. It allows developers to comment directly on issues and commits. This keeps coversations attached to the associated code changes and increases awareness.

GitHub's popularity and novelty has sprung a number of studies on its use and the implications for developers in OSS projects. GitHub employs a DVCS model. Previous studies have found DVCSs can change development processes [3] and influence developer behavior [30]. Qualitative studies have determined that GitHub's social coding features are used by OSS developers to follow others' activity [12], [13], [25] and make decisions to participate [27] and accept or reject contributions from others [42]. Quantitative studies have looked into the structure of the GitHub user environment [39], [40], the complex effects of social media on OSS project success [43], and the implications of adopting DVCS models for the development process [16].

These studies have focused only on OSS projects. OSS teams are structured on the premise that distribution is their only feasible way of interaction [9]. They employ practices like open access to information, visibility of developer activity, and self-governance [10], and they seem to be more resilient to the strains of distributed work. Traditional commercial projects operate very differently [15]. Tasks are assigned and developers are often not aware of other changes being made outside of their area of code [44]. These practices make commercial projects more prone to collaboration challenges like communication breakdowns, a lack of awareness, and code conflicts. Some research has investigated how commercial projects can adopt OSS-like practices to mitigate these challenges [31], [36]. However, many open questions remain. The benefits of transparency, a key feature of GitHub, has been said to be an important avenue for research in commercial projects [36]. Further, the impact of the adoption of OSS-like practices on collaboration in commercial projects has yet to be studied.

In this paper, therefore, we examine how **commercial organizations with projects producing proprietary software** use GitHub for their collaborative software development. Our study systematically analyzes the different aspects of collaboration that typically are identified as challenging in the literature of collaborative software development as reviewed above; communication, coordination, awareness, task division and conflict resolution. Our study was driven by the following research question:

> **RQ:** What practices do commercial software projects follow in conjunction with GitHub's features to collaborate? What is the effect on their collaboration?

## III. Study design

We used a mixed-method approach in a qualitative study to examine collaborative practices of projects from commercial organizations using GitHub. Through an initial survey sent to 1,000 GitHub users, we collected responses from 240 participants. The survey polled participants about their use of GitHub, reasons for adopting GitHub and the effect it has had on their development practices. From the pool of survey respondents that volunteered for interviews we selected those that indicated they participate in collaborative projects. We conducted in-depth, semi-structured interviews with the first 30 respondents on our list. After the first interviews we realized that several interviewees used GitHub primarily as part of their job. This gave us a great opportunity to look into a type of projects that has not been investigated yet, although we had to forfeit the chance to triangulate our results since we did not distinguish between commercial and OSS-related use in the survey. Out of our 30 interviews, 24 came from commercial projects and 6 from OSS projects.

The 24 interviewees from the commercial organizations are part of development teams that use GitHub without making their code publicly available. They do that through private repositories in GitHub's web-based interface or by utilizing GitHub's enterprise version which they host on their own servers. Throughout the paper we will refer to the interviewed cases as "commercial projects" or "commercial organizations", and use the following definition, in accordance with other studies (e.g. [17]), for the term "commercial" in our analysis and discussion of findings:

> In the context of this paper we use the term "commercial" to refer to projects or organizations that produce proprietary software and do not make their software visible in public repositories or receive contributions by entities outside the commercial organization.

To answer our research question, we analyzed the data from all 30 interviews. To distinguish the practices of the commercial projects we explicitly compared the information from the 24 commercial project interviews with those gathered through the other, 6 OSS project interviews.

### A. Survey procedure

We acquired a list of recently active users through GitHub's public API. The primary goal of the survey was to recruit participants for interviews; it put us in touch and gave us an initial feel of our audience. We kept the survey short and the questions straightforward to keep our respondents interested in proceeding with an interview. Our survey consisted of open-ended questions. We asked participants why they adopted GitHub ("What was the reason you decided to use GitHub?") and how it has affected their development process ("How has the use of GitHub affected your development process?"). We also included one closed-ended question which asked participants if they use GitHub primarily to collaborate with others or for solo projects. At the end of the survey, we asked participants whether they would volunteer for an interview with us.

We piloted the survey internally with members of our research group and externally with 100 GitHub users. We obtained 19 responses (19% response rate) and conducted trial interviews with four of these survey respondents. We used these pilot survey responses and interviews only to fine-tune the wording of the questions of our survey and build our interview script; their results are not included in our analysis. After this pilot phase, we sent a revised survey to 1000 GitHub users and received 240 responses (24% response rate).

### B. Survey responses

We used thematic analysis techniques [8] to process the survey responses and understand our participants before proceeding with getting data from the interviews. We gave the participants open-ended questions and coded their answers into exclusive categories. Table I contains a summary of the coded survey responses. The label "N/A" corresponds to unanswered questions. The majority of the respondents use GitHub primarily to collaborate. The most common reasons noted by our respondents for adopting GitHub were the desire to contribute to OSS projects and prior experience with git.

The most cited effect on development practices from the use of GitHub is adopting a workflow through branching and pull requests. The participants also reported writing better code as a result of using GitHub, due to the self-consciousness that public visibility creates. Adoption of best practices was another effect; this included code reviews, frequent commits, and generally learning from looking at others' code. Respondents also noted writing more code and contributing more as a result of using GitHub. 13% of the respondents attributed whatever effect they saw in their development to using DVCS, not GitHub specifically.

### C. Interview procedure

Eighty two survey respondents volunteered for follow-up interviews. We selected the first 30 volunteers and conducted one-hour long semi-structured interviews. All interviews were recorded and transcribed to facilitate iterative analysis.

TABLE I. CODED SURVEY RESPONSES SUMMARY.

| **Survey item** | **Survey responses** | **240** |
|---|---|---|
| GitHub use | Primarily for collaboration | 148 (62%) |
| | Primarily for solo projects | 90 (37.5%) |
| | N/A | 2 (0.5%) |
| Reason for adopting GitHub | To contribute to OSS or share code | 67 (28%) |
| | Because they used git already | 52 (22%) |
| | To collaborate with others | 35 (15%) |
| | To host projects and store files | 26 (11%) |
| | GitHub's popularity | 24 (10%) |
| | GitHub's interface / ease of use | 24 (10%) |
| | GitHub was adopted at work | 8 (3%) |
| | Other reasons | 4 (2%) |
| GitHub's effect on development | Adopt branching workflow | 53 (22%) |
| | Be conscious about writing better code | 44 (18%) |
| | Adopt/learn best practices | 43 (18%) |
| | Write more code / Contribute | 41 (17%) |
| | Same effect as using any DVCS | 31 (13%) |
| | N/A | 28 (12%) |

To focus our interview questions, we started all interviews by asking the interviewee in what setting they primarily used GitHub (job-related or not). We then asked the interviewees' to describe their current job, their responsibilities, their team, and their typical workflow. We guided the interviewees to provide details about how they interact with other members of their team, the tools they use daily as part of their collaboration, and any challenges they face. In the remainder of the interview, our questions focused on the collaboration elements: *Communication, coordination, awareness, task division and conflict resolution*. We asked how their team handled each element and how their use of GitHub impacts each element. The questions we used for this part of the interview are shown in Table III.

Similar to our survey responses, we used thematic analysis techniques [8] in processing interview data. We loosely grouped our data around the collaboration elements and performed open coding, in which we assigned hierarchical codes to interesting parts of the interviews. The code system developed iteratively, by coding, discussing codes, combining and splitting codes, and writing memos about more abstract phenomena as they emerged. We then used axial coding to iterate over the data we had collected, which allowed us to build the answer to our research question.

Table II summarizes some basic characteristics of the interviewees.

TABLE II. INTERVIEWEES' AND PROJECTS' DESCRIPTIVE CHARACTERISTICS.

| | **Interviewees** | **30** |
|---|---|---|
| GitHub use | Primarily for job-related projects | 24 (80%) |
| | Primarily for OSS contribution | 6 (20%) |
| Job situation | Professional developers | 25 (83%) |
| | Managers/CTOs (also do development) | 4 (13%) |
| | Developers in internship | 1 (4%) |
| | **Job-related projects** | **24** |
| Distribution | Distributed | 16 (67%) |
| | Collocated | 8 (33%) |
| | **Median team size** | **7** |

TABLE III. INTERVIEW QUESTIONS CORRESPONDING TO COLLABORATION ELEMENTS AND HOW GITHUB SUPPORTS THEM.

| Collaboration element | Sub-area | Question |
|---|---|---|
| Coordination | Definition | What does coordination mean to you? How would you define it? |
| | Example | Can you give an example of coordination in your team? |
| | Coordination Needs | What are the occasions that you find it essential to coordinate with your team? |
| | Issues/Solutions | Can you remember an issue with coordination? How did you resolve it? Do you have any conventions on coordination you follow in your team? |
| Task division | Criteria | How do you decide how to split the work between team members? |
| Awareness | Activity | How do you keep track of activity in your project? |
| | People/Artifacts | Do you prefer to track the activity of people or artifacts? Why? |
| | Maintaining Awareness | How do you stay aware of actions or changes in the artifacts? |
| | Challenges/Problems | Does it get too much? Is something missing? |
| Communication | Communication needs | What are the occasions that you find it essential to communicate with your team? |
| | Challenges/problems | Does it get too much? Is something missing? |
| Conflict resolution | Conflicts | Do you come across conflicts? How do you resolve them? |
| GitHub's support | Role | How does GitHub fit in with all those collaboration pieces? |
| | Benefits | How has GitHub helped your team collaborate? |
| | User evolution | Has your use of GitHub changed over time? How? |
| | Problems | Is there something missing? Does GitHub hinder anything? |

Out of the 30 interviewees, 24 reported on *the use of GitHub in commercial projects*; 20 were professional developers and 4 were managers or CTOs that also participate in the development activities. The 6 participants that indicated using GitHub primarily for OSS projects also identified themselves as professional developers (5 out of 6) with the exception of 1 developer in internship. The majority of the commercial projects that interviewees were part of were geographically distributed (16 out of 24).

The *commercial project* interviewees are part of small development teams, the median team size in our group of interviewees was 7. Even in the cases that developers are working for large organizations running multiple projects with development teams having more than 10 members, they reported that a standard practice is to form sub-teams with a maximum of 4-5 developers working on any given project. When asked, these interviewees indicated that they are also active contributors to other, OSS projects, either by maintaining their own public repositories or by contributing code to others' repositories.

## IV. FINDINGS

**RQ:** What practices do commercial software projects follow in conjunction with GitHub's features to collaborate? What is the effect on their collaboration?

We based our study on the interview responses of 30 participants that use GitHub for collaborating with others, either in a commercial setting or for OSS projects. In this section we present our findings. We draw the answer to RQ based on the data from the 24 interviews about commercial software projects. In parallel we compare the answers of commercial projects to answers coming from our 6 OSS interviews.

The reported collaborative development practices from commercial projects focus on four underlying themes: enhancing *independent work* by using a branching workflow, reducing *communication and coordination* by using GitHub's visibility, *self-organizing* for task assignment and conflict resolution, and *using external tools* when interacting with non-technical members and project management.

### A. Working together through independent work

Commercial projects on GitHub use a workflow that builds on branching and pull requests to drive independent work.

GitHub offers two different development models (based on Git) resulting in different workflows. In the *fork & pull model* collaborators keep their own local copy (a "fork") of the repository and make changes there, issuing a pull request when done to inform the project maintainer about the new changes. Upon review the changes are pulled in the original repository. In the *shared repository model* team members have direct push access to the repository.

In our interviews, we asked participants to walk us through the workflow that their team is using. 23 out of the 24 interviewees reported that their team is using a workflow using pull requests. This pull-based workflow was reported in two flavours: a simple one (19 interviewees) and a complex one (4 interviewees), described below.

We graphed the workflow to which 19 out of the 24 (79%) interviewees converged, shown in Figure 1. This workflow uses a central repository that belongs to the organization and then developers work on features on separate branches within the repository. After the developer is finished with committing their changes to their local branch they submit a pull request, making their changes visible to the rest of the team for code review. Four interviewees reported using a more complex variation of the workflow in Figure 1 that uses branches for development, fixes and releases in addition to the feature branches, therefore helping with release management [2].

The remaining interviewee reported that his team is using a typical centralized workflow heavily influenced by SVN but using git commands (shown in Figure 2), as they had recently migrated and were not familiar with all of git's workflow capabilities.

The major difference between the workflows in Figures 1 and 2 is the use of pull requests.

[2]all four cited the branching model at nvie: http://nvie.com/posts/a-successful-git-branching-model/

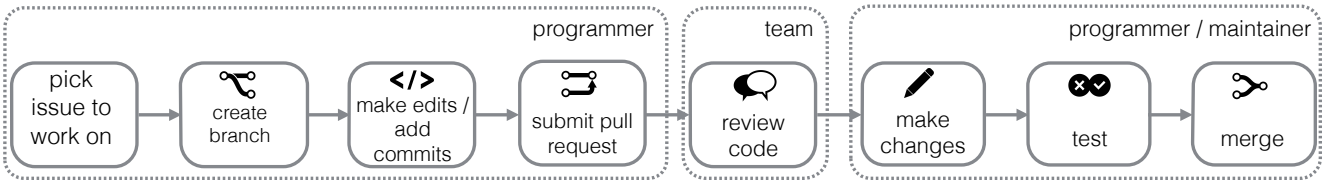


Fig. 1. Branching, pull-based workflow used in 79% of interviewed cases. Another 17% used a more complex variation of this type of workflow.

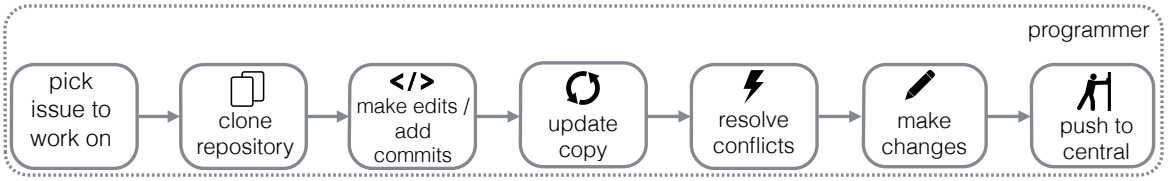


Fig. 2. Centralized workflow reported by one commercial team in our study, using git commands instead of SVN.

> *"When I first started we all pushed to one branch, but the problem is you push and nobody knows what changes are going through and there is no chance for a review. Our current system is that you don't make any changes without submitting a pull request[...]and then one or more members will review it and you need one thumbs up from another team member to merge it, otherwise it can't go in. "* [P13 - professional developer in commercial project]

GitHub provides two collaborative development models resulting in two workflows: *fork & pull* and *shared access* [3]. In the fork & pull model developers create copies of the original repository (called "forks") to which they commit their changes, using them as their independent isolated workspace. Developers submit a pull request to signal they have changes in their fork and, usually after inspection, those can be integrated in the original repository by a project maintainer. The fork & pull model is popular with OSS projects for reducing coordination requirements. In the shared repository model team members have direct push access to the main repository, a model considered suited to small teams and organizations [3]. However, surprisingly almost all the small teams and organizations that we interviewed reported using a hybrid between the two models, a *branch & pull* workflow. This workflow allows having one repository for the project (as in the shared access model) but isolate feature development within it through branches, and still use pull requests (as in the fork & pull model).

Pull requests were reported to have a dual utility: they signal coordination points, and they give the opportunity for code review. Looking at Figure 1 we have highlighted the activities that fall under different roles, as the interviewees reported them from their practice, where individual work is intermittent with team work. A submitted pull request is the last action before the hand-off of the new contribution to the team to discuss and review and, therefore, a clear (and critical) point when coordination is needed. The result of the code review can indicate needed changes; in that case the programmer makes edits and adds commits and creates a new pull request that will go through code review. Upon reaching agreement either the contributor or the maintainer take the final actions before merging. In the branching workflow shown in Figure 1 there is explicit provision for the team to weigh in before new code is merged and that lowers the risk of skipping code review. That is in contrast to the centralized workflow the interviewees reported (Figure 2) where the developer works entirely in isolation before they push their changes and they show up in the central repository.

We should note that contrary to git being a DVCS supporting workflows that build on each developer having a complete copy of the repository and history as their individual development environment, this was not the way it was used in the commercial projects we interviewed.

We see that commercial projects on GitHub use pull requests in the same way as OSS projects do. Previous GitHub studies have shown that pull requests are used in OSS projects roughly as much as shared access to the repository [16]; they provide an opportunity for code review when the pull request is received [16], and they work as a screening mechanism when open source projects receive contributions from new participants [25]. This was also confirmed by our OSS interviewees:

> *"Even if you are not sure if the other dev is capable of contributing good code, you can review pull requests and if the fifth pull request is good you give him/her commit bit."* [P3 - professional developer and OSS project founder]

> *"In the case of Fedora, if anyone is making a significant commit to a project, even if they have commit access because they are on the team, they still open a pull request and the code isn't merged in unless somebody gives it a +1. So, we use the pull requests to review code before we merge it in."* [P16 - professional developer in OSS project]

In our case, interviewees from commercial projects did not mention explicitly using pull requests as a screening mechanism; we assume that this is because there is already established familiarity or trust between team members due to their existing experience working together.

[3]https://help.github.com/articles/using-pull-requests/

Commercial projects on GitHub use pull requests in the same way as OSS projects do; to isolate individual development and perform code review before merging.

### B. Reduced communication and coordination needs

Commercial projects use GitHub's visibility to focus their communication and coordination needs; mostly on questions and problems during code reviews and merges.

Table IV shows the interviewees' preferred source of information to track the progress and status of the project. For the most part, GitHub's visible, automated output is preferred to avoid superfluous communication. Programmers reported getting the information they need through the issue list to see outstanding items, and the commits list to see ongoing activity. Notification emails were another option mentioned, although interviewees are conscious about information overload. IM was reserved for keeping interviewees aware of blockers; IM clients that integrate with GitHub (the most cited example was Campfire [4]) combine update messages with ongoing communication, used as needed without being intrusive. Interviewees commented that updating the status of issues so that the newsfeed is current requires effort, but preferred this to having additional communication for awareness.

TABLE IV. INTERVIEWEES' PREFERRED METHOD OF KEEPING TRACK OF ACTIVITY.

| | **Awareness source** | |
|---|---|---|
| Progress / Status | Issue list | 6 (25%) |
| | Commit list | 7 (29%) |
| | Notification e-mails | 5 (21%) |
| | Chat client integrating with GitHub | 6 (25%) |

Interviewees recognized two needs for communication taking place on GitHub: questions for other team members when encountering problems, and discussions around specific coding items.

> *"As far as collaboration and communication, on GitHub most of the communication and collaboration has to do with one of two things: either concerns, or the aftereffects of a change."* [P6 - professional developer in commercial project]

Interviewees reported directing their questions to team members by using the @mention functionality on GitHub as their first line of communication (13 out of 24 interviewees – 54%). GitHub has implemented the @mention convention as a notification trigger, sending an email to a user that was mentioned relative to an artifact, drawing their attention and focusing communication as a result. Code-centric communication was reported to be successfully handled through GitHub comments: inline comments for code reviews, and issue, commit, and pull request comments for communication surrounding a specific artifact. For more elaborate conversations, however, the majority (20 out of 24 interviewees – 83%) reported moving to external tools such as an IM client, mailing lists, or having direct communication when applicable. The communication venues outside GitHub afforded the teams more space and synchronicity to discuss ideas about the software, compared to discussing particular fixes through comments on GitHub. The switch between different communication tools for different purposes is a communication protocol that teams have grown into through continuous use, which helps team members know where they would find information or communication relevant to a particular subject.

All interviewees consider coordination essential for collaborative development. Their fear was that inadequate coordination can lead to duplicated work due to lack of awareness of others' activity, while too much coordination was seen as overhead. They found that a highly visible project status mitigates the danger of an awareness gap when development is happening independently, and replaces the need for communication and coordination. Also, a branching strategy dictates the steps of the process the team needs to follow and limits coordination needs to specific points, as was mentioned earlier with pull requests.

Similar use of GitHub as a communication platform was reported from OSS interviewees too.

> *"Through github specifically there is not really a messaging system, so everything is down on a commit or a pull request or something like that. We don't really use github as a direct communication thing unless directly talking about bits of code and pull requests. We usually find that IRC or email works better for side notes. Like I said you can't send a PM on github."* [P16 - professional developer in OSS project]

Dabbish et al. [12] have reported similar findings for OSS projects on GitHub. They found that comments are used by OSS projects for direct feedback and code review, and that activity information flows across the site in the form of updates. These findings agree with what is already known for OSS projects in general [48].

Commercial projects use GitHub's transparency as OSS projects do; their communication is code-centric and pull requests act as a coordination mechanism.

### C. A touch of self-organization

Commercial projects use self-organization when assigning tasks and resolving conflicts.

Commercial project interviewees reported using self-assignment to divide tasks between them, based on their expertise and availability. This practice was reported by 16 out of the 24 interviewees (67%).

> *"Mostly we self-assign tasks over the sprint planning process, usually a person who has experience with the application will volunteer more so than someone who doesn't. Sometimes there is a task that is very very specific to an issue and one or two developers have worked on it in the past and they will take it up."* [P6 - professional developer in commercial project]

In the cases of self-assignment of tasks project managers and/or team leads still participate in the estimation

[4] https://campfirenow.com/

and prioritization of tasks, to define them to be "bitesize" [P14 - professional developer in commercial project], but then developers pick which tasks they will work on. In 54% of the cases (13 of 24 interviewees) the team leads were the ones that followed the task assignment decisions the developers made to maintain awareness.

The reported benefit of self-assignment was that since team members know their expertise, if they self-organize to define and select tasks to work on, the team is collectively working with the optimum task division. Interviewees pick tasks from GitHub's issue list (only 6 interviewees – 25%) or the bug tracker, task management or project management tool they are using that integrates with GitHub but is an external tool. The most used tools respectively were JIRA [5], Asana [6], and Pivotal Tracker [7]. The reason for not using GitHub's issue tracker was that interviewees found it too minimal for their teams' needs.

> *"Pivotal gives us a lot of really great features and allows very fine grained control over the process of the ticket, every status of it along the lifecycle is kept track of. GitHub has more of an open-closed status."* [P34 - professional developer in commercial project]

The second area that involved emergent decisions was resolving conflicts. Interviewees use external continuous integration tools for tests and builds; these show where problems are and then decisions are up to the team on whether to solve the problem individually or collaboratively.

> *"We use a CI server and when it sees an update it pulls the code down, runs the test suite and if it's all green it will rebase that up to the master branch. If the test cases are failing we work together to try and see why it failed, or we send it back to whoever broke the build because you can't merge that code."* [P24 - professional developer in commercial project]

Self-organization is a long known practice in OSS projects [9]. In fact self-assignment of tasks has been pointed out as a difference between how OSS projects and commercial projects handle task-actor dependencies [11]. From our evidence we see that the practices may not be that different in some cases.

| Commercial projects on GitHub practice task self-assignment similar to OSS projects. |
|---|

### D. Challenges in collaborating with non-technical members

| Commercial projects resort to using external tools when collaborating with non-technical members operating outside GitHub. |
|---|

One of the challenges that the commercial projects reported about using GitHub was occasions when they have non-technical members on their team that they have to interact and work with. Team members that are not programmers are reluctant to use GitHub as they find it complicated. This is acknowledged by developers, but adds to the set of tools they need to use and the sources of information they need to keep track of so that they don't overlook a request.

> *"The CMO, the other officers in the business, pretty much anyone who's not technical, they would use Asana for logging feature requests. That's a thing about GitHub, it could be hard to find things, and I would never ask a non-technical person to use GitHub. Never."* [P5 - manager/developer in commercial project]

While there is no coping strategy reported to deal with this issue, the measure that is taken is the use of a separate issue tracker or chat room dedicated to non-technical users, where they can report bugs or post feature requests. Although non-technical users would be expected to find dealing with version control tools difficult, it is surprising that they find it challenging even for reporting issues through GitHub's interface.

Another challenge area for programmers is the intersection between their activities and project management, which most of the times creates additional coordination and communication requirements to fulfill the management's needs for monitoring the project's progress. This is a less severe problem when at least one member of the management team is also a programmer, but that is reported as the exception rather than the rule.

> *"I associate coordination with hassle and work that has to be done, because that's what I remember, the frustrating part about coordination. How do big open source projects just manage themselves somehow? It's not that simple, I know, because there are developers even there who make the primary decisions, but because the developers are speaking the same language it is easier."* [P19 - professional developer in commercial project]

## V. Discussion

Throughout the previous section we reported how commercial development teams use GitHub in practice. We answer our research question by summarizing our findings in Table V, mapping each practice that our interviewees reported as helping their smooth collaboration to the corresponding GitHub feature that enables it. The third column shows the effect on the corresponding collaboration element as it was related with each practice by our interviewees.

The practices that we observed small commercial teams following in our study are not new. They have been known and used in OSS projects, as shown by other GitHub studies that present findings from OSS projects, as well as OSS-related literature. We summarize the comparison between the practices reported for commercial projects using GitHub and OSS projects in Table VI. To avoid repetition we have only included the first column from Table V. We give examples of where the same practices have been reported for OSS projects; from our own 6 OSS interviewees or from previous studies of OSS projects, using GitHub or not.

GitHub served as an enabler for adoption of OSS-style practices in the teams in our study, making their collaborative

[5] https://www.atlassian.com/software/jira
[6] https://asana.com/
[7] http://www.pivotaltracker.com/

TABLE V. PRACTICES REPORTED BY COMMERCIAL PROJECTS, THE CORRESPONDING GITHUB ENABLERS, AND HOW THEY SUPPORTED THE COLLABORATION ELEMENTS IN OUR STUDY

| **Reported practice** | **GitHub enabler** | **Collaboration element effect** |
|---|---|---|
| Independent development | Branching workflow (also mentioned as GitHub workflow) | Minimized *coordination* needs |
| Progress and status monitoring | Timeline, notifications, integration with chat client | *Awareness* |
| Code reviews | Pull requests with in-line comments | Highlighted *coordination* needs |
| Code-centric communication | Comments on commits, issues, and pull requests | Targeted *communication* |
| Self-organization | Public issue tracking | *Task Division* |
| Automatic testing and deployment | Service hooks in corresponding tools | *Conflict Resolution* |

TABLE VI. OVERLAP BETWEEN REPORTED COMMERCIAL PROJECT AND OSS PROJECT PRACTICES.

| **Reported practice** | **Also in OSS projects** |
|---|---|
| Independent development | P3, P16, [16] |
| Progress and status monitoring | [12], [13], [25] |
| Code reviews | P16, [16], [32] |
| Code-centric communication | P16, [12], [18] |
| Self-organization | [9], [10], [11] |
| Automatic testing and deployment | [30] |

development process smoother and efficient as we discuss in Section V-A below. This is not the only recorded occasion of organizations adopting OSS-style practices for their proprietary software development, however. Literature shows that large organizations are interested in promoting open collaboration by opening project participation to all employees in the organization and use OSS-style collaborative development practices to do so, a trend termed Inner Source [37]. In Section V-B we discuss how GitHub's characteristics and mode of adoption can support open collaboration inside commercial organizations.

### A. GitHub as a vehicle for adopting best, OSS-style, practices

For some software organizations (usually small in size) the primary focus is on making their daily operations of collaborative development more effective. That means looking for solutions and best practices for their development and collaboration. Our evidence showed that GitHub is a viable option as a CDE in small software companies; they made use of the pull-based workflow to enhance code reviews and GitHub's integration of tools and information, typically found in CDEs.

First, the adoption of a pull-based workflow for a company's proprietary software projects was the most surprising finding in our study. So far OSS projects have used the fork & pull development model to have control over the quality of the contributed code through incremental code reviews and screening new contributions [16], [42]. GitHub offers an easy-to-use implementation of pull requests and their use as a code review mechanism through in-line comments to commercial projects too. Besides the fork & pull model, GitHub also supports a development model where all collaborators have direct commit access to a main repository. For small teams inside companies, where development is restricted to a predefined group and members have established trust through prior collaboration, the shared repository development model seems more fitting (asserted also by GitHub [8]). However, the teams we studied used a hybrid between the two models: the team maintained one repository to which all collaborators had access but made the decision to use pull requests between branches.

This *branch & pull* variation has advantages over both the fork & pull and shared repository development models, in a company setting. The fork & pull model assumes multiple forks of the original repository. A team member or a project manager needs to view all the individual forks if they want to see ongoing activity, which makes it less traceable than using multiple branches that are visible in the main repository. At the other end of the spectrum, the simple shared repository model gives team members direct commit access. It still leaves traces of activity after commits have been made but less distinctive than a pull request. The teams in our study recognized better traceability of changes and the ability to do code reviews as the benefits of pull requests over the simple shared repository model; both benefits are especially important for development teams that continuously deploy and ship code.

Second, GitHub serves as a CDE that integrates tools and information under the same interface. Leveraging the visibility of information, team members can have focused, artifact-based communication. We saw development teams in software companies using GitHub's visibility of repository activity information to their advantage regarding their communication and coordination needs. Maintaining awareness through issue lists, commit lists and notifications was the first line of defence against communication overhead. Such practice is supported by GitHub's transparency and is in line with how information needs are typically facilitated in OSS projects [12], [18]. Some communication needs cannot be fulfilled by just the visible information however. In the case of questions and problems for example, team members want to contact others. The development teams in our study used GitHub's @mention notification trigger to draw other members' attention, and kept communication close to the corresponding artifact through comments. More detailed conversations were handled through external tools, like IM clients that integrate with GitHub and show committing activity. Although interviewees didn't mention breakdowns, future work can look into whether the use of external tools has negative side effects for transparency, team knowledge management and discovery. The commercial teams we studied practiced self task-assignment; this is not a GitHub-specific capability but can still function because team members view all updates regarding who is working on what.

[8] https://help.github.com/articles/using-pull-requests/

### B. GitHub as a vehicle for open collaboration in companies

Organizations (usually large in size) try to facilitate open collaboration within the company by following OSS-inspired practices, a trend referred to as Inner Source [37]. Open collaboration, in an organization's context, means encouraging participation in software development projects to anyone within the organization irrespective of the team or department they belong to [31]. Seeing benefit in inter-team collaboration within the organization, companies seek to reproduce the emerging and open collaboration conditions of OSS projects. *The goal is to bring and keep together disparate employees or teams, mimicking communities of volunteers around the company's projects*. To facilitate this goal organizations turn to the way OSS communities form around projects and what communication and workflow patterns they follow [2]. This should not be confused with companies doing their software development in public, producing OSS software, or collaborating with contributors from outside the organization's boundaries. Below we argue that GitHub is a viable solution as an open collaboration tool within the confines of an organization looking to use OSS-style development and collaboration practices for its proprietary software development.

Research has shown interest in the potential of replicating OSS-style practices within commercial organizations building proprietary software. A few studies have tried to synthesize the processes that have proven effective for OSS projects and how these could be implemented and benefit traditional organizations in a more general way. Feller & Fitzgerald [15] proposed a framework to analyze and understand OSS development, recommending further, systematic studies if the software industry was to successfully enact OSS principles. More recently, Stol et al. [37] developed a framework that outlines factors that support the successful adoption of Inner Source in organizations. The framework was developed through analysis of the literature examining the adoption of OSS-style practices in companies running proprietary software projects, and was empirically validated through three case studies. The adoption factors cover characteristics of the software product, the development methods and forming a company-wide community around projects. Other practitioner reports and case studies have presented cases of organizations that have tried mimicking the behaviour of OSS projects for their proprietary development; examples include HP [14], Lucent Technologies [17] and Nokia [23].

While this literature shows that some progress has been made in studying the adoption of OSS-style practices in commercial projects, a number of open questions remain. Two areas that have not been fully investigated in commercial projects are the benefit of transparent development environments [37] and the problems of achieving standardization of tools and practices [31]. Below we discuss our findings, which clearly indicated OSS-style practices in commercial projects using GitHub, in light of these two open issues.

*1) Transparency makes centralized information visible:*

> GitHub can support open collaboration within commercial organizations by centralizing tools and information and making them transparent to all potential collaborators.

When organizations begin to adopt open collaboration they open participation to all members of the organization. Assuming such an initiative is successful the number of collaborators on projects can increase and, thus, introduce communication and coordination overhead. This overhead can be mitigated by using tools, such as GitHub, that promote visibility of development work to create a more transparent development environment. It has been shown, in OSS projects, that transparency of the workspace can reduce communication since developers are aware of others' behavior [12], [13]. However, research has not yet investigated how transparency can benefit commercial projects looking to adopt open collaboration [37].

In our study, we saw that GitHub's transparency was beneficial to commercial projects in several ways. First, project activity is made visible through the commit and issue lists as well as by receiving notifications. This centralization and visibility of all project and developer activity allowed team members to obtain information about the progress and status of the project and maintain awareness. This awareness helped reduce communication overhead. By following the traces of activity recorded on GitHub, teams limited their communication to problems and questions not easily answerable from the existing information. They used GitHub's code-centric communication medium of comments and/or lightweight, visible, text communication in a way similar to OSS-like projects [18]. The visibility of project activity also supports open collaboration since potential new collaborators can see ongoing code changes and past design decisions.

GitHub's transparency also manifests in the use of the pull-based branching workflow. The commercial projects in our study uniformly showed a preference for the pull-based workflow (Figure 1) which makes the new contributions (and their information traces) more visible than the shared access, centralized model. The pull requests act as a coordination mechanism for code review. This supports open collaboration by allowing new collaborators to easily participate in a code review since all information is attached to the pull request.

Teams in our study also used the transparent, integrated environment of GitHub as a ground to practice self-organization, one of the most well-known principles of OSS projects. Self task-assignment was supported by having one central, integrated space of activity and information, where developers can see updates about open and closed tasks and what other members are working on. The same has been recognized as a requirement for open collaboration to function within an organization's boundaries [23], [31].

*2) Agents of change towards a de facto tool and process:*

> The adoption of GitHub can bring OSS-style practices in an organization, with the help of peer interaction, and support open collaboration.

Open collaboration in an organization is hindered by the lack of common processes and tools between teams. Having to use different development environments when moving between teams or projects can introduce delays and barriers to contribution for collaborators. Previous work has found that organizations, although they recognize the potential benefits, are unable to obtain standardization across projects and teams [31].

While standardization may be difficult, social interaction can mediate the problem. Previous work has shown that peer interaction, in the form of recommendations and observations, can lead to tool discovery and adoption [28], [29], [47]. Our findings agree with this. Developers who already use GitHub and work in commercial projects act as *agents of change*, recommending the use of GitHub in the workplace too. While this may start with individual developers and teams, it can end up gaining ground within the organization in a subversive manner. We saw cases in our study where the final decision to adopt GitHub came from the bottom up or from the top down.

> *"The reason that we moved to GitHub was because I was using it personally and so at that point I approached the company and I said that we are using git and it'd be nice to have this functionality."* [P28 - professional developer in commercial project]

> *"I decided we should use GitHub and part of it is picking a tool that a lot of people are going to be familiar with"* [P21 - CTO in commercial software organization]

GitHub comes with an embedded process which was used consistently by our interviewees. Even though multiple workflows are supported in GitHub, the commercial projects we interviewed converged towards essentially the same pull-based workflow, already known to be in use in OSS projects [16]. Even more, GitHub has formed a community around its service that due to its openness makes common and best practices visible to all. Even if companies are not using or participating in the GitHub social coding environment for their proprietary software development, they can still observe it. One of our OSS interviewees remarked:

> *"I think it's really interesting how all the projects collaborate on there and I think when companies move their code to GitHub they become aware of other projects that are on GitHub and they pay attention to how those projects do things and so everybody who host their project on GitHub eventually starts to use the tool in almost the same way and they realize the benefits of the social coding aspect with all the comments and the pull requests and stuff like that."* [P16 - professional developer in OSS project]

This quote shows that observation can also lead to adoption and use of a tool or practice, as shown by previous studies [28]–[30], [47]. As the GitHub environment itself is heavily influenced by OSS development and collaboration practices, it spreads OSS-style processes to commercial projects too. Through the consistent use, both the tool and the process can become established *de facto* standards in an organization that have the potential to support open collaboration.

## VI. Threats to validity

Our study faces the threats to validity that qualitative studies face. A construct validity threat is the interviewees' understanding of the concepts of collaboration and coordination, especially given their code-centric perspective. We mitigated this risk by asking interviewees for examples and by providing explanations when needed, to ensure a mutual understanding of the constructs. A further construct validity threat comes from the fact that GitHub is built on top of git and there is unavoidably significant in their functionality. Our interviewees may not have made an explicit distinction; on our part we specifically focused our questions on GitHub and believe that our results are valid despite the overlap.

In terms of internal validity, the participants to our survey and interviews were active but self-selected, and we can only rely on what they report given their time and motivation to participate in our study. Finally, in terms of external validity, we achieved saturation in the results of the participants we studied but the themes, principles, and practices we found may not generalize to everyone. The majority of the participants we interviewed are professional, experienced developers which makes us confident that our insights are valuable and relevant.

## VII. Conclusion

In this paper, we presented a study exploring the use of GitHub by software development teams in commercial organizations. The number of commercial organizations using GitHub for their private, proprietary software development is growing fast and practitioners are interested in learning how GitHub's features are used by other companies for collaborative software development [49]. To the best of our knowledge, this is the first study that looks at how commercial projects use GitHub. Our paper makes the following contributions:

- ***It brings evidence of how GitHub is used in commercial projects.*** In our study, commercial software projects adopted a pull-based workflow as a development model and took advantage of doing code reviews on pull requests, in a manner consistent with peer review in OSS projects. While this development process can surely be attributed to git and DVCS in general, GitHub also provided awareness through visibility and kept communication and coordination focused and lightweight. GitHub is a good vehicle for organizations that are looking to adopt current best practices in their development. These results can be useful for companies and teams that are looking for ways to introduce current best practices in their collaborative development or are specifically considering adopting GitHub. Future research can look into how widespread and beneficial the use of pull requests is in commercial projects by looking at a larger number of companies.
- ***It adds to the literature studying the adoption of* OSS*-like practices in organizations.*** We discussed the role that GitHub can play in fostering open collaboration within a corporation. Firstly, GitHub can serve the purpose of becoming a single place for tools and information within an organization, similar to other CDEs. Also, given its popularity, GitHub enters organizations organically through developers acting as change agents. Its consistent use makes it a de facto common tool and process that enables transparent, de-coupled work across teams. This addresses open questions in the literature regarding the role of transparency and lack of standardization in organizations that aim for open collaboration.


## Acknowledgments

We kindly thank all the participants of the study for taking the time to provide us with their insight. This work was partly funded by NSERC Canada.

## REFERENCES


[1] F. Abbattista, F. Calefato, D. Gendarmi, and F. Lanubile. Incorporating social software into distributed agile development environments. In *Automated Software Engineering - Workshops, 2008. ASE Workshops 2008. 23rd IEEE/ACM International Conference on*, pages 46–51, Sept 2008.

[2] J. Bacon. *The Art of Community: Building the New Age of Participation*. O'Reilly Media, 2009.

[3] E. T. Barr, C. Bird, P. C. Rigby, A. Hindle, D. M. German, and P. Devanbu. *Cohesive and Isolated Development with Branches*, volume 7212 of *Lecture Notes in Computer Science*, pages 316–331. Springer Berlin Heidelberg, 2012.

[4] W. Bourne. 2 reasons to keep an eye on github. http://www.inc.com/magazine/201303/will-bourne/2-reasons-to-keep-an-eye-on-github.html, 2013.

[5] M. Cataldo and K. Ehrlich. The impact of communication structure on new product development outcomes. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, pages 3081–3090, New York, NY, USA, 2012. ACM.

[6] M. Cataldo and J. D. Herbsleb. Coordination breakdowns and their impact on development productivity and software failures. *IEEE Trans. Software Eng.*, 39(3):343–360, 2013.

[7] M. Cataldo, J. D. Herbsleb, and K. M. Carley. Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity. In *Proc. of the 2nd ACM-IEEE Int. symposium on Empirical software engineering and measurement*, pages 2–11. ACM, 2008.

[8] J. Corbin and A. Strauss. *Basics of qualitative research: Techniques and procedures for developing grounded theory*. Sage Publications, 3rd edition, 2008.

[9] K. Crowston, K. Wei, J. Howison, and A. Wiggins. Free/libre open-source software development: What we know and what we do not know. *ACM Comput. Surv.*, 44(2):7:1–7:35, Mar. 2008.

[10] K. Crowston, K. Wei, Q. Li, U. Y. Eseryel, and J. Howison. Coordination of free/libre open source software development. In D. E. Avison and D. F. Galletta, editors, *ICIS*. Association for Information Systems, 2005.

[11] K. Crowston, K. Wei, Q. Li, U. Y. Eseryel, and J. Howison. Self-organization of teams in free/libre open source software development. *Information and Software Technology Journal: Special issue on Understanding the Social Side of Software Engineering, Qualitative Software Engineering Research*, 49:564–575, 2007.

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

[13] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb. Leveraging transparency. *IEEE Software*, 30(1):37–43, 2013.

[14] J. Dinkelacker, P. K. Garg, R. Miller, and D. Nelson. Progressive open source. In *ICSE '02: Proceedings of the 24th International Conference on Software Engineering*, pages 177–184, New York, NY, USA, 2002. ACM.

[15] J. Feller and B. Fitzgerald. A framework analysis of the open source software development paradigm. In *Proceedings of the Twenty First International Conference on Information Systems*, ICIS '00, pages 58–69, Atlanta, GA, USA, 2000. Association for Information Systems.

[16] G. Gousios, M. Pinzger, and A. v. Deursen. An exploratory study of the pull-based software development model. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 345–355, New York, NY, USA, 2014. ACM.

[17] V. K. Gurbani, A. Garvert, and J. D. Herbsleb. A case study of open source tools and practices in a commercial setting. *SIGSOFT Softw. Eng. Notes*, 30(4):1–6, May 2005.

[18] C. Gutwin, R. Penner, and K. Schneider. Group awareness in distributed software development. In *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work*, CSCW '04, pages 72–81, New York, NY, USA, 2004. ACM.

[19] J. D. Herbsleb. Global software engineering: The future of socio-technical coordination. In *FOSE '07: 2007 Future of Software Engineering*, pages 188–198, Washington, DC, USA, 2007. IEEE Computer Society.

[20] B. K. Kasi and A. Sarma. Cassandra: Proactive conflict minimization through optimized task scheduling. In *Software Engineering (ICSE), 2013 35th International Conference on*, pages 732–741. IEEE, 2013.

[21] E. Knorr. Github's new ceo: We're serious about the enterprise. http://www.infoworld.com/t/application-development/githubs-new-ceo-were-serious-about-the-enterprise-249524, 2014.

[22] F. Lanubile, C. Ebert, R. Prikladnicki, and A. Vizcaino. Collaboration tools for global software engineering. *Software, IEEE*, 27(2):52–55, 2010.

[23] J. Lindman, M. Rossi, and P. Marttiin. Applying open source development practices inside a company. In *OSS2008: Open Source Development, Communities and Quality (IFIP 2.13)*, volume 275/2008 of *IFIP International Federation for Information Processing*, pages 381 – 387. Springer, 2008/// 2008.

[24] T. W. Malone and K. Crowston. The interdisciplinary study of coordination. *ACM Comput. Surv.*, 26(1):87–119, 1994.

[25] J. Marlow, L. Dabbish, and J. Herbsleb. Impression formation in online peer production: Activity traces and personal profiles in github. In *Proceedings of CSCW '13*, pages 117–128, New York, NY, USA, 2013. ACM.

[26] J. McAnally. This week on github: In good company. http://www.linux-mag.com/id/7348/, 2009.

[27] N. McDonald and S. Goggins. Performance and participation in open source software on github. In *CHI '13 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '13, pages 139–144, New York, NY, USA, 2013. ACM.

[28] E. Murphy-Hill, R. Jiresal, and G. C. Murphy. Improving software developers' fluency by recommending development environment commands. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, pages 42:1–42:11, New York, NY, USA, 2012. ACM.

[29] E. Murphy-Hill and G. C. Murphy. Peer interaction effectively, yet infrequently, enables programmers to discover new tools. In *Proceedings of the ACM 2011 Conference on Computer Supported Cooperative Work (CSCW)*, CSCW '11, pages 405–414, New York, NY, USA, 2011. ACM.

[30] R. Pham, L. Singer, O. Liskin, F. Figueira Filho, and K. Schneider. Creating a shared understanding of testing culture on a social coding site. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 112–121, 2013.

[31] D. Riehle, J. Ellenberger, T. Menahem, B. Mikhailovski, Y. Natchetoi, B. Naveh, and T. Odenwald. Open collaboration within corporations using software forges. *IEEE Software*, 26(2):52–58, 2009.

[32] P. C. Rigby, D. M. German, and M.-A. Storey. Open source software peer review practices: A case study of the apache server. In *Proceedings of the 30th International Conference on Software Engineering*, ICSE '08, pages 541–550, New York, NY, USA, 2008. ACM.

[33] A. Sarma, G. Bortis, and A. van der Hoek. Towards supporting awareness of indirect conflicts across software configuration management workspaces. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, ASE '07, pages 94–103, New York, NY, USA, 2007. ACM.

[34] A. Sarma, L. Maccherone, P. Wagstrom, and J. Herbsleb. Tesseract: Interactive visual exploration of socio-technical relationships in software development. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, pages 23–33, USA, 2009. IEEE Computer Society.

[35] A. Sarma, D. F. Redmiles, and A. van der Hoek. Palantir: Early detection of development conflicts arising from parallel code changes. *IEEE Transactions on Software Engineering*, 38(4):889–908, 2012.

[36] S. Sharma, V. Sugumaran, and B. Rajagopalan. A framework for creating hybrid-open source software communities. *Inf. Syst. J.*, 12(1):7–26, 2002.

[37] K.-J. Stol, P. Avgeriou, M. A. Babar, Y. Lucas, and B. Fitzgerald. Key factors for adopting inner source. *ACM Trans. Softw. Eng. Methodol.*, 23(2):18:1–18:35, Apr. 2014.

[38] M.-A. D. Storey, D. Čubranić, and D. M. German. On the use of visualization to support awareness of human activities in software

development: a survey and a framework. In *Proceedings of the 2005 ACM symposium on Software visualization*, SoftVis '05, pages 193–202, 2005.

[39] Y. Takhteyev and A. Hilts. Investigating the geography of open source software through github. http://takhteyev.org/papers/Takhteyev-Hilts-2010.pdf, 2010.

[40] F. Thung, T. Bissyande, D. Lo, and L. Jiang. Network structure of social coding in github. In *17th European Conference on Software Maintenance and Reengineering (CSMR)*, pages 323–326, 2013.

[41] J. Tsay, L. Dabbish, and J. Harbsleb. Let's Talk About It: Evaluating Contributions through Discussion in GitHub. In *FSE '14: Proc. of the 22nd Int. Symp. on Foundations of Software Engineering*, Nov. 2014. To appear.

[42] J. Tsay, L. Dabbish, and J. Herbsleb. Influence of social and technical factors for evaluating contribution in github. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 356–366, New York, NY, USA, 2014. ACM.

[43] J. T. Tsay, L. Dabbish, and J. Herbsleb. Social media and success in open source projects. In *Proc. of the ACM 2012 conf. on Computer Supported Cooperative Work Companion*, pages 223–226. ACM, 2012.

[44] P. Vitharana, J. King, and H. Chapman. Impact of internal open source development on reuse: Participatory reuse in action. *J. Manage. Inf. Syst.*, 27(2):277–304, Oct. 2010.

[45] J. Whitehead. Collaboration in software engineering: A roadmap. *Future of Software Engineering*, 0:214–225, 2007.

[46] S. Wong, Y. Cai, G. Valetto, G. Simeonov, and K. Sethi. Design rule hierarchies and parallelism in software development tasks. In *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*, pages 197–208, Washington, DC, USA, 2009. IEEE Computer Society.

[47] S. Xiao, J. Witschey, and E. Murphy-Hill. Social influences on secure development tool adoption: Why security tools spread. In *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing*, pages 1095–1106, New York, NY, USA, 2014. ACM.

[48] Y. Yamauchi, M. Yokozawa, T. Shinohara, and T. Ishida. Collaboration with lean media: How open-source software succeeds. CSCW '00, pages 329–338, NY, USA, 2000. ACM.

[49] N. C. Zakas. Github workflows inside of a company. http://www.nczonline.net/blog/2013/05/21/github-workflows-inside-of-a-company/, May 2013.