The Code-Centric Collaboration Perspective: Evidence from GitHub

Eirini Kalliamvakou
University of Victoria
ikaliam@uvic.ca

Daniela Damian
University of Victoria
danielad@cs.uvic.ca

Leif Singer
University of Victoria
lsinger@uvic.ca

Daniel M. German
University of Victoria
dmg@uvic.ca

ABSTRACT

Decentralized Version Control Systems (DVCS) enable teams to structure their development independently, allowing parallel work. Online hosting services for DVCS have become a viable option for teams to structure their workflow around, and GitHub is currently the most widely used such service, hosting over 10 million[1] repositories for open source and proprietary software projects. What are the mechanisms through which the use of DVCS-based workflows supports a team's collaboration? In our study we decompose collaboration into its elements of coordination, communication, awareness, task division, and conflict resolution. We provide qualitative evidence of teams using GitHub benefitting in terms of collaboration, and explain the benefit by seeing how each collaboration element is supported. We discover that using DVCS goes hand-in-hand with a code-centric collaboration perspective inspired by self-organization and mediating communication and coordination needs.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management—*Programming teams*

General Terms

Management

Keywords

Decentralized Version Control, Git, GitHub, Workflow, Collaborative development

1. INTRODUCTION

The choice of development practices has implications for the collaboration process of a team, and even though certain decisions are technical in nature, they have organizational impact. For example, Version Control Systems (VCS) target the management of the codebase, but go hand-in-hand with the team's collaboration process by highlighting how contributions from team members are handled. VCS provide a basis for the rules and steps team members are to follow when managing, reviewing and integrating changes to project artifacts. The resulting workflow outlines how team work is organized [23] and pinpoints when and what interactions are needed between team members, framing collaboration.

Decentralized Version Control Systems (DVCS) enable development in a more independent manner. Team members have their own clones of a project and use branches to work on tasks in parallel to other members, minimizing the touch points between them. By moving to a peer-to-peer configuration, DVCS provide an alternative code management model, resulting in their own workflows [7], but the mechanisms through which collaboration is influenced are not studied. Our goal is to understand what a software team's choice to use DVCS translates into for its collaboration.

Currently, Git is the most recognizable DVCS and has gained momentum by providing workflows that make it easy for developers to work in isolation, as well as collaborate by integrating their work with their peers' when ready. Developers without write-access to the original repository signal they have changes in their clone through a pull request and, after inspection, those can be integrated by a project maintainer. Also, light branching and merging enables developers with write-access to still separate their activities by purpose, in a dedicated workspace, and merge frequently.

GitHub is a web-based code hosting service for projects using Git, building on its functionality. It provides code-hosting in public or private repositories, and technical and social features to support collaborative development. Studies on GitHub have focused on how its features are used to assess developer and project properties [13, 21, 33], or how they influence development practices [17, 24]. While development benefits are clear, the more subtle implications for collaboration have not been explicitly studied.

Our qualitative study aims to describe and understand collaboration in teams following DVCS-driven workflows, using GitHub as a proxy. We decompose collaboration into elements, tracking where problems usually appear, as per the literature. We observe a code-centric collaboration view in the teams we studied, and the mediation of collaboration problems. Abstracting from our findings, we explain how DVCS workflows minimize collaboration challenges overall by

[1] https://github.com/blog/1724-10-million-repositories

FSE Submission 2014, Hong Kong

specifically targeting each collaboration element. Our findings include data coming from software organizations, with activity not limited to public repositories.

2. BACKGROUND & RELATED WORK

Collaboration of software teams is made up of members' interdependent actions, coordinated towards a common goal [34]. We follow Malone & Crowston's [20] view of coordination as the direct or indirect actions needed to manage those interdependencies. We synthesize a team's **collaboration** process by observing how it handles **coordination** through **task division**, **awareness**, **communication**, and **conflict resolution**. We refer to these as *the collaboration elements*.

Coordination and communication involve known challenges that result in failures and longer resolution times [4, 6, 5, 19]. Maintaining awareness of interdependencies is also challenging; tools can provide alerts of potential coordination needs and recommend communication between interdependent developers [27, 29], but it is not without overhead [3]. Conflicts occur even in the presence of awareness tools and teams spend effort to resolve them. Finally, although modularization of tasks is an accepted way to minimize interdependencies [35], it is impossible to remove them altogether.

OSS teams seem to be more resilient to the strains of distributed work because they are by default structured on the premise that distribution is their only feasible way of interaction [10]. Coordination mechanisms based on self-governance [11], a set of simple communication tools [18], and a task-based meritocratic culture [36] are enablers for effective production of quality software even on a large scale. Research has looked for ways to transfer some of the benefits from the OSS to the commercial development environment by proposing frameworks that build on the structure, processes and culture of OSS teams [30, 26].

There has been a growing number of studies focused on the use of GitHub and its implications for developers. The addition of social features to GitHub's code hosting services has been the focus of qualitative studies, investigating how these features are used to assess activity and potential in projects [13, 14, 21, 33, 22]. Quantitative studies have looked into the structure of the GitHub user environment [32, 31] and the implications of adopting DVCS models for the development process.

GitHub (building on DVCS) is enabling a different development process, managing code contributions through a pull-based model [17], with identified benefits in terms of process automation, and separating the building of artifacts from the integration to the codebase. The resulting workflow is a process where team members are isolated during the coding activities and interact on context-specific occasions during integration. We anticipate that this has implications for the collaboration elements, e.g. the amount of coordination and communication needed to take place in the team.

We, therefore, aim to answer the **research question**

How do DVCS*-based workflows support collaboration?*

by looking at how each collaboration element functions under these workflows and whether the known collaboration challenges persist. Making the use of GitHub our proxy also allows us to learn more about how teams use its features to shape or support their collaboration process.

3. STUDY DESIGN

As the issue of how the collaboration process functions in teams using DVCS and its resulting workflows has been largely unexplored, we chose to conduct an exploratory, qualitative study. Team collaboration is complex to capture and greater than the sum of tools and steps, so wealth of information is critical. We designed our study to start with a survey with open-ended questions to shape our initial observations, followed by interviews with 35 survey participants for in-depth investigation of their collaboration practices. Throughout our study, we iteratively analyzed the data we collected in a process based on Grounded Theory [9].

We based our initial curiosity on previous work, which had shown that decentralized version control models can change development processes [1] and that interfaces on top of DVCS, such as GitHub, can influence developer behavior [24]. As Git is currently the most popular DVCS and GitHub is the largest service hosting Git projects, we chose to use GitHub as our proxy. Through GitHub's public API, we acquired a list of recently active users in May 2013.

3.1 Survey procedure

Based on our interest in how collaboration happens in projects that have adopted DVCS, we created a survey with open-ended questions. The goal of the survey was to act as a recruitment tool for interviews, getting us in touch and giving us an initial feel of our audience. We, therefore, kept the survey short and the questions straightforward to keep our respondents interested in proceeding with an interview. Their answers could also be used as "subject hooks" in case we had difficulties getting interviewees to provide detailed input. We asked participants about why they adopted GitHub, whether they use it to collaborate, how it helps them collaborate, how they use GitHub to track activity of others, and how GitHub has affected their development process. At the end of the survey, we asked participants whether they would volunteer for an interview with us.

After piloting the survey with members of our research group, we invited 100 GitHub users to answer the survey as an external pilot. This led to 19 responses (19% response rate) and four pilot interviews. We adjusted our survey after the pilot by adding a question about dependencies between team members to gauge team coordination practices. We sent our revised survey to 1000 GitHub users. For this main survey, we received 240 responses (24% response rate).

3.2 Interview procedure

We created an interview script to conduct semi-structured interviews, allowing us to pursue additional insights. Through the responses to the external pilot and the pilot interviews, we were made aware of the fact that some participants used GitHub for their job, while others used the tools for personal, OSS projects only. We used the opportunity to get insight from those using GitHub primarily for job-related development on how GitHub is used in a commercial setting, through activity in private repositories. Based on what we learned from the pilot, we therefore adjusted our interview script to account for the distinction between job-related and personal-interest projects. From those participants who volunteered for interviews, we decided to talk to 35 of them. All interviews were recorded and then transcribed to facilitate iterative analysis. On average, an interview lasted one hour. The interview questions were formulated around our

main proxy question:

How does GitHub support collaboration?

The first part of the interview started with general questions about what the interviewee used GitHub mostly for, in what setting (job-related or not), and what type of repositories. We then asked about descriptive information as applicable: job situation and responsibilities, company size, team size and geographical distribution. Finally, we asked interviewees to describe their typical workflow including interactions with their team and provide an example, the tools and technologies they use daily as part of their collaboration, and any challenges they face.

Next, we asked questions relating to how the team handles each of the collaboration elements. Our goal was to gather as much information as possible about how each team collaborates, and therefore specifically focused on GitHub only at the end of this part of the interview. At that point we asked interviewees to reflect on what is GitHub's role in their collaboration, how it supports it for them, and what benefits or challenges they have noticed. We have included the questions used for this part of the interview in Table 1 below. The complete set of questions can be found [here].

We followed a systematic approach to organize the unstructured data from the interviews, based on the principles of Grounded Theory for processing qualitative data [9]. We loosely grouped our data around the collaboration elements and we started our analysis with open coding, in which we assigned hierarchical codes to interesting parts of the interviews. The code system developed in an iterative process, in which we coded, discussed codes, combined and split codes, and wrote memos about more abstract phenomena as they emerged from the data. We then used axial coding to iterate over the data we had collected, which allowed us to build an answer to our research question.

3.3 Participants

Our survey had 240 respondents with a 24% response rate. 148 respondents (62%) said they use GitHub primarily for collaboration, and 90 (38%) said they use it primarily for solo projects. We coded the answers to the survey question about reasons for adopting GitHub, leading to the categories in Table 2. The interview results are presented in Section 4.

To contribute to OSS or share code	67 (28%)
Because they used git already	52 (22%)
To collaborate with others	35 (15%)
Just as storage space	26 (11%)
Because of GitHub's popularity	24 (10%)
Because of GitHub's interface / ease of use	24 (10%)
The adoption of GitHub was decided at work	8 (3%)
Other reasons	4 (2%)

Table 2: Respondents' reasons for using GitHub

Most interviewees use GitHub primarily for team projects related to their job (about 68%), some use it mainly for maintaining their own or contributing to others' OSS projects (about 18%), and only few use it exclusively for hosting personal projects. 28 interviewees (82%) explicitly said they use GitHub to collaborate with others. Of these interviewees, 25 (71%) were professional developers, 4 (11%) were developers and / or managers, 3 (9%) were students, and 3 (9%) were developers in internships. Overall, even though we targeted users of an OSS-centric website, the majority of our interviewees were professional developers and focused on how they use GitHub to collaborate as part of their job. This also allowed us to inquire in depth about the work processes used in private, commercial projects using GitHub's infrastructure—a valuable detail normally hidden from public view.

The interviewees are part of small development teams, mostly 4-8 developers. Even in the cases that developers are working for large organizations running multiple projects with development teams having more than 15 members, they reported that a standard practice is to form sub-teams with a maximum of 4-5 developers working on any given project. All interviewees are active with OSS projects, either by maintaining their own public repositories, or by contributing code to others' repositories.

In the remainder of the paper we present our findings (Section 4), abstract from them (Section 5.1), provide an answer to our research question (Section 5.2), and discuss its origin (Section 5.3) and implications (Section 5.4).

4. RESULTS

In this section we report the evidence that emerged through analyzing the interviewees' responses, as triggered by their quotations. The input we gathered focuses around four areas: the interviewees' *perspective on collaboration*, which provides the frame of interpretation for their answers, the adoption of GitHub's *pull request functionality* to organize their work, the relaxed *communication and coordination requirements* of teamwork as a benefit of using GitHub to collaborate, and the *challenges* that interviewees have encountered when using GitHub.

4.1 Working together through independent work

We found that the interviewees hold a code-centric view of development work and the collaboration that goes into it; their view is focused on and bounded by the code artifacts.

Although there were cases where the interviewees also had managerial roles that included talking to clients, making decisions on the project's direction, or picking new features for implementation, the vast majority of the participants were only involved in the programming activities. Their perspective and experience is, therefore, focused on the implementation activities that start after planning and high-level task division is concluded, and end with deployment to production. The most common starting point of their workflow was a list of features entered in one form of a ticketing system or another, and its subsequent breakdown to issues entered in GitHub's issue tracker.

Interviewees view development as a problem-solving activity fulfilling a specific task, and do not focus on the business logic behind it or the value that it will offer the client. Development activities that fall within this area were attributed to management, analysts, or even designers, and were not recognized as the concern of programmers or addressed through using GitHub. In alignment with this code-centric perspective, the interviewees' view of collaboration is that of a setup that allows to easily share and integrate code coming from separate sources when needed, but otherwise allowing for independent coding with the minimum possible requirement for coordination either between programmers, or between programmers and management.

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	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?

Table 1: Interview questions corresponding to collaboration elements and how GitHub supports them.

4.2 Pull requests to commit, merges to coordinate

We found that pull requests not only provide a development model, but also highlight times when coordination is needed. Git, combined with a branching strategy and workflow to manage contributions, allows interviewees a high degree of independent and parallel coding, and GitHub helps them organize their work, on the individual and team level.

Pull requests were the feature mentioned most frequently by the interviewees, used for two purposes: as a *screening tool* and access model when there are contributions from less experienced or unknown developers, and as an *alternative commit model* to encourage code reviews earlier rather than debugging later. Almost all the interviewees that use GitHub as part of their job, reported that their company uses a combination of private and public repositories. Private repositories are used for hosting the code that is part of the project's commercial advantage, while parts of code are published as libraries or gems and openly available to any developer interested in using or extending them. Almost all interviewees using GitHub to contribute to OSS projects reported that contributions are almost exclusively handled through pull requests.

> *"Shared repository is great if you have really high confidence in the people who have access to it, or people that you now have trust with. I would prefer to have a sort of probationary period with pull requests."* [P5]

Interviewees handling access rights for the development team talked about access to the main repository as an issue of trust in a contributor's programming skills. In most cases the primary repository is the place from where code is pushed to production and some form of gatekeeping is required, which is what repository forks and pull requests support. Team members maintain their own fork of the main repository, work on changes there, and then submit a pull request when ready. This allows for less experienced developers' code to be filtered before added to the production branch, either through testing or based on GitHub's signal for whether the code is going to be a clean merge or not. The same principle applies to incoming contributions from volunteers who are unknown to the team and want to contribute to its public repositories.

> *"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]

Even in the cases when all team members' code is trustworthy, interviewees said that they are in favour of adopting the pull requests as their commit model. Issuing a pull request signals the time for code review right when the code is ready and combined with unit testing and continuous integration allows work to be performed faster and automatically. Interviewees recognized incorporating the process of doing code reviews upon receiving a pull request as a best practice, one they are in the process of discussing with their team or already adopted because they started using GitHub.

> *"If things continue on in a feature branch and diverge rather than merge back to that known good GitHub point, the consequence is that it gets harder to coordinate when it's time to merge."* [P25]

Doing code reviews upon receiving a pull request was reported as a best practice by interviewees. Merges are the coordination points that call for developers' attention, and adopting a practice of code reviews and frequent commits, is seen by the interviewees as minimizing the coordination effort, since doing code reviews on smaller pieces of code is reported as easier than debugging bigger code chunks later.

Also, in the occasion of merge conflicts, reverting to an older, stable version results in the least possible wasted/lost effort, because of the short time window between releases.

4.3 Minimized communication and coordination needs

We found that interviewees utilize the visibility GitHub offers as well as pull requests to minimize and target their communication and coordination actions. Recognized needs were questions and problems during code reviews and merges.

The interviewees agree that awareness of the progress and status of the project is important, along with possible blockers coming from the work of other team members or subteams. However, they are skeptical about the added overhead of communication and coordination to fulfill this awareness need.

> *"Everybody has the opportunity and is directly made aware through notifications of all changes to the codebase. We never had a situation where someone went "wow, where did this new stuff come from" or "wow, who broke my code" because everybody is coordinated because of these tools."* [P16]

The status of the project is visible through GitHub's issue tracker, or any similar tool that integrates with GitHub, as well as the commit list, and developers can stay updated through notification emails. Interviewees are conscious about information overload due to a high volume of notifications coming from large, active projects, although the small team size helps to keep the volume manageable. As far as blockers are concerned, communication tools like IM clients or chat rooms serve the purpose of ongoing communication, used as needed. These tools are unobtrusive and provide an added means of keeping up-to-date as they provide notifications that relate to issues or commits as incoming messages without interrupting a programmer's work. Interviewees commented on the conscious effort required to update the status of issues so that it is current and visible to everyone, but viewed this as a more desirable process rather than having additional communication targeted towards awareness.

> *"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]

The major recognized needs for communication are questions that developers have for other team members when encountering problems (tackled sufficiently through chat, e-mail, or direct communication when applicable), and discussions around a specific coding item. The latter communication need is successfully handled through comments on GitHub, on either issues, commits, or pull requests. This code-centric communication is seen as an enabler for faster and better development by the interviewees, and they consider it as an integral part of their work and learning process, and not added overhead.

Another practice that moderates the amount of communication and coordination needed is following a practice of self-assignment regarding dividing tasks between team members. Interviewees collectively agreed that when development activities are handled between the programmers, there is less friction when working and collaborating. The development team members know their knowledge level and expertise, and if they can use that criterion when defining and selecting tasks to work on, the team is collectively working with the optimum task division. The cases when the development team's activities intersect with project management or other teams or departments create communication and coordination challenges, as discussed later on.

It should be noted that the interviewees commented on self-organization as a successful practice to be adopted from the OSS world. They recognized two thorny organizational issues for self-management to work effectively, however. One is that management has to trust developers with having control over what they work on, which is something that is met with reluctance. Secondly, the right recruitment is essential, where the weight is on getting self-motivated individuals that function within a group by sharing the same goal.

All interviewees consider coordination essential to smooth collaborative development. Their major concern with inadequate coordination is that it can lead to duplicated work due to lack of awareness of others' activity, while too much coordination is seen as overhead. The actions discussed earlier for maintaining a highly visible status of the project mitigate the danger of an awareness gap at the stage when development is happening independently. At the same time, a branching strategy and commit model incorporated in the team's workflow dictates the steps of the process they need to follow and limits coordination needs to those points. Pull requests signal merging points and coordination needs for actions on doing code reviews, testing and merging the code back to the master branch using specific tools and process, once developers are alerted.

Interviewees overall reported very smooth joint action. Code conflicts are rare when it comes to merges, mostly because the process of code review and GitHub's functionality allows for conflicts or errors to be spotted easily and early. Conflicts between team members only rarely occur and they are related to code style disagreements or suggestions for alternative solutions to the programming problem at hand. On the occasions that there were serious conflicts that resulted in difficulties for the project, or delays, those were of a higher-level, organizational nature (e.g the direction of an OSS project) or disagreements with members of management lacking technical knowledge (e.g management insisting on the use of specific tools or having unrealistic expectations).

4.4 Challenges with non-technical members

We found that interviewees didn't report challenges in using or adopting GitHub within the development team. Their reported challenges involved interaction with non-technical members that operated outside GitHub.

> *"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]

> *"We have 2-3 biologists, they are used to things like Dropbox and sharing files through e-mail. It would be great if they created a ticket on GitHub and I wouldn't have extra communication through*

e-mail with them. On the other hand I can appreciate that the GitHub interface for biologists, for outside people, non-developers, isn't optimal." [P9]

One of the challenges that the interviewees 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, a fact that is acknowledged by developers. However, the problem remains as it adds to the set of tools that programmers need to use and the sources of information they need to keep track of so that there is no loss of awareness or a request is not overlooked. 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.

"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]

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. In most cases, the lack of technical background of managers puts strain on the communication with developers in an attempt to get insight on the rationale behind technical decisions. The management's non-technical expertise occasionally also results in having unrealistic expectations of developers, a fact that manifests as tension during communication.

4.5 Summary

Overall, the interviewees' view of collaboration departs from the more traditional view that requires or assumes that team members are working jointly on something [15]. The programmers' view of collaboration is consisting of tasks that are performed independently until there is need for them to intersect or merge, restricting the need for coordination and communication only to those occasions. This independent action is supported by DVCS and workflows that follow a clear process of committing, reviewing, testing, and deploying code. Additionally the coordination and communication requirements of merge points can be fulfilled easily by the use of simple tools. GitHub serves exactly this programmer perspective by providing the combination of features needed so that, by their account, developers can fulfill all the tasks without creating organizational overhead. Most interviewees reported that they have or are willing to change their collaboration practices to better utilize GitHub's functionality and feature set. Most of them reported that their team arrived at their current collaboration process through trial and error (or that they are still experimenting) but those that are following the workflows and branching strategies enabled by Git and GitHub are experiencing faster and smoother collaboration.

In Table 3 we summarize the practices that our interviewees put forward as helping their smooth collaboration process as they have been enabled by the corresponding GitHub feature. We discuss further on how each collaboration element is supported by each practice, answering our research question, in Section 5.2.

5. DISCUSSION

In this section we add context to our findings by characterizing and analyzing them. Firstly, we have extracted three major themes, common across our interviewees, representing the principles behind their code-centric perspective: Focus, independence, and exposure. Secondly, we synthesize the findings and themes to answer our research question. Thirdly, we interpret our findings in light of work management and the decision making that takes place on different levels of organization. Finally, we pose some open questions that form implications for further research based on our current findings.

5.1 Identified themes

5.1.1 Focus

Our interviewees are sensitive about distractions from their main coding activities. *Focus* represents targeted needs fulfilled by the minimum set of tools and actions.

Simple tools. The interviewees described situations where their teams have experimented with different tools as part of their collaboration process, but are either considering or have decided on using few and simple ones, aligning their collaboration process with GitHub's functionality. Their view has shown that as long as the development and committing process is clear and adhered to, they do not recognize a need to use multiple tools and they have a preference towards configurations that integrate functionality in a single location. GitHub offers them a number of workflow variations, owing to Git's branching and merging rules for managing the codebase, incoming contributions, and releases. Bug reports and feature requests are handled through simple issue tracker tools while communication is sufficiently carried out in IM chat clients and/or chatrooms.

The simplicity of the tools does not appear to compromise the teams' ability to maintain awareness of the work that is carried out and using that as a mediator for coordination needs. Gutwin et al. [18] similarly observed that the use of simple, visible, text communication was adequate in enabling developers of OSS projects to maintain good awareness of other developers' activities, a finding that seems to also cover progress awareness, as evident in our case. Although in [18] work artifacts were not represented in the communication system, our interviewees consciously choose to use communication tools that integrate with GitHub (such as Campfire, which was the most frequent example) and allow for events such as merges, commits etc to appear as incoming messages. This highlights an attempt to adopt communication principles from the OSS environment to teams developing commercial software as well.

Focused interaction. All programmers in our study have shown affinity to communication that is focused around a specific concern, activity, or artifact. They are frugal when

Code-centric collaboration practice	Collaboration element effect	GitHub enabler
Independent development	Minimized *coordination* needs	Forking repositories and branching
Clear, concise process of committing	Minimized *coordination* needs	git flow (also mentioned as GitHub workflow)
Visible progress and status	*Awareness*	Timeline, notifications, integration with chat client
Code reviews	Highlighted *coordination* needs	Pull requests
Code-centric communication	Targeted *communication*	Comments on commits, issues, and pull requests
Self-organization	*Task Division*	Public issue tracking
Automatic testing and deployment	*Conflict Resolution*	Service hooks in corresponding tools

Table 3: Code-centric collaboration practices supporting collaboration elements through GitHub

it comes to coordination activities that don't serve a specific purpose at hand, and most reported negative experiences with general purpose meetings, even more so with colleagues or teams outside the development team. GitHub is regarded as a good venue for code-centric, focused communication that is asynchronous and unobtrusive, especially through commenting on any type of artifact. At the same time, the option of having real-time communication or discussions that are documented is open through the use of chat or mailing lists. GitHub acts as the place of record for all code-related communication, as well as the results of discussion that has taken place elsewhere.

GitHub highlights when there is need to coordinate activities between members. The tagging functionality alerts developers when they have been mentioned in connection to an issue or commit, drawing their attention when necessary to act upon something. The submission of pull requests signals the need for a developer to review the code and merge it. The fork & pull model is therefore utilized in a stigmergic manner [2] similar to that observed in OSS projects, by generating traces of activity that inform other developers' work and minimize, as well as focus, coordination needs.

5.1.2 Independence

Interviewees appreciate *Independence* as the de-coupling of their work from others', which allows them to self-organize.

Decentralized work. The use of decentralized version control allows developers to work independently, which was the most frequently cited reason our interviewees and their teams decided to use Git or GitHub, as well as the major benefit they have experienced from doing so. The capability of having their own local copy of the repository allows them to work on changes, fixes, or features on their own, without being restricted by someone else's work or having to be necessarily connected to a network to do so. Unblocking each developer's work from the obligation to wait for others' commits or to checkout the latest version (as is the case with centralized versioning), means that the speed of development is maximized. This freedom is again reminiscent of OSS projects where developers are strongly independent in their contribution behaviour.

A key factor to the independence of team members' work is the visibility of activity. Dabbish et al. [13, 14] highlighted the transparency of the workspace as the major driver for observing information rather than obtaining it through communication, in much a similar way to the principle of stigmergy [2]. Transparency helps support independence of development by supplementing decentralized work with the clear picture of the project that is necessary for individual development effort to not be redundant or duplicated.

Low need for management. All interviewees shy away from rigid management or team structure and favour self-organization when it comes to selecting what to work on and how to proceed with development. At the same time, a clear, concise, and well-known workflow and development process is recognized as essential. Having a process but also the freedom to self-organize allows developers a level of independence and keeps them motivated, similar to OSS projects where the role of management is usually low. The interviewees that were team leads or CTOs also confirmed that it is their goal that management is as light as possible and that it takes more the form of a source of support for developers when they run into problems, rather than deciding what they should work on, when, or how. Those with managerial responsibilities said that their major goal is to make developers happy and try to recreate the feeling of working on something they like with their friends ("a developer's wonderland") which is the best motivation for them, much stronger than deadlines or metrics.

Developers do recognize the need for management to monitor progress of the activities in the project and to make estimations on its completion. A desirable state reported by the interviewees was one where at least one member of the management or business team is educated in using GitHub, so that they can enter requests and/or track the status of development without creating communication or coordination overhead.

5.1.3 Exposure

Given the interviewees' strong interest in collaboration, *Exposure* allows them to easily share work with others and pushes them to produce better code.

Easy contribution process. Git and GitHub provide developers with an easy way to contribute to a codebase. OSS projects are highly discoverable because of GitHub's wide use and mostly first place in search engine results. Once discovered, all project content is highly visible and its progress is reflected in the issue tracker, commits, etc. Developers can then proceed to fork a project, work on it locally, and submit a pull request with their changes, a process that has been recognized as lowering the barriers to contributing [22].

For commercial projects as reported in our study, the same ease of use benefits the work taking place in public repositories, where external contributions can range from correcting a typo in a wiki page to contributing a patch or commenting. This maximizes the exposure that the organization and the commercial project can receive while it capitalizes on commits that come from developers outside the team.

Peer pressure. Due to the exposure and visibility offered through GitHub, developers, although inclined to contribute more, are more conscious of the quality of the code they submit. Interviewees contributing to OSS projects reported that in many cases they will have their code in a private repository for some time working on improvements before they make it public. In commercial projects, developers are also more careful because there is more visible ownership of code, much like in OSS projects, and it is easy to detect when someone commits code that introduces a bug or is of low quality. Along those lines, teams also adopt the practice of smaller, more frequent commits, as mentioned earlier. It is, therefore, possible that this visibility indirectly results in higher quality software being produced as also observed in [13].

In Section 4 we observed how teams of developers use GitHub and DVCS in practice and the related benefits they have experienced. The practices and workflow they follow revealed a shifted collaboration perspective, with the identified themes above representing the principles on which it operates. Below we interpret how and why these principles work, looking at conditions of the collaborative environment.

5.2 How do DVCS-based workflows support collaboration?

The code-centric perspective our interviewees have expressed targets the core of collaboration. To developers, **collaboration is equivalent to managing independent contributions to the common whole**. While individual contributions are to be worked on in a decentralized, independent way, a workflow consisting of clear and concise steps to accept, review, and integrate them is crucial to contribution management. This de-coupled sense of collaboration appears to result in a collaborative behaviour that benefits the elements of collaboration.

Instead of spending effort in maintaining awareness of every developer's actions or all changes made, to ensure that the subset they need to be aware of is covered, our interviewees opt for **awareness of the project's progress and status through visibility**. In such a transparent environment [14], developers can follow the information they deem relevant, without communication overhead or management, and in an unobtrusive manner. To maintain this visible state, however, all team members are individually responsible for signalling the status of their work.

In a workflow that supports decentralized work, **coordination needs are minimized and focused on merges**, as points requiring joint action. Lower coordination needs also require less communication as the vehicle. This does not mean that the team loses information flow or rapport between its members. **Communication is code-centric and focused on specific artifacts**, and is archived to provide record of decisions and solutions. The benefit is communication and coordination that is smooth and efficient because it is targeted.

The decentralized workflow principles allow conflicts to be spotted at code review before merging or handled automatically through testing and continuous integration environments. This transforms **conflict resolution from a group concern to an individual task** that is picked up when the process followed highlights the need to do so. Due to self-assignment and ownership of tasks, the code changes needed to resolve conflicts are by default the responsibility of the original developer. Self-assignment based on expertise optimizes the collective performance of the group, since members work on tasks or fix bugs in a way that capitalizes on their skills. The support for each collaboration element is summarized in Table 3, answering our research question.

The combination of a workflow as a structured process for the team, and maintaining awareness through visibility, agrees with the concept of continuous coordination [25], a paradigm under which work practices are both structured and flexible. Structure is translated into a specified protocol for managing code changes and providing synchronization checkpoints of independent work. Flexibility builds on transparency of work, enabling self-organization. Although continuous coordination is assumed to be served by sophisticated tools [28], our evidence suggests that the principles of continuous coordination can be achieved by the use of much simpler tools, urging us to rethink their contribution relative to the power of process.

The application of the principles we identified for viewing and practicing collaboration appears to simplify the collaboration process. This is an important effect when we consider how expensive it is (in terms of both time and money) to set up teams and experiment with the combination of tools and practices that allow them to collaborate efficiently. The reported evidence points towards the benefits that might be expected if the perspective of collaboration is shifted.

5.3 Decision levels and OSS-like mentality

For the rest of our discussion, we look at the development project or organization as a stratified structure of organizing work and making decisions, similar to the Curtis et al. layered behavioural model [12], adapted as shown in Figure 1. At the basis of this structure is the operational layer, concerned with decisions and tasks that are highly technical in nature and focus on the implementation of solutions. In self-organizing programmer teams the technical decisions can be individual in terms of the best course for problem-solving, but also involve the rest of the team members when it comes to use of specific tools and processes for their collaboration. One level up is the managerial layer, focused on the description of problems and the design of solutions, looking at the project as a whole. The managerial layer is also responsible for decisions that enable the operational layer to function efficiently by having the resources and processes it needs. Depending on the size of the team, company or organization, an additional layer might be in place above the managerial one, concerned with strategic decisions regarding the vision and direction of the team within the organization, or having a more holistic view of the organization and its function and position within a broader corporate environment.

The evidence we have collected comes from developers that are part of the operational level, and concerned with the nuts and bolts of implementing the decisions of the level above them. At the operational level, therefore, we have observed that there is a collaboration perspective at play where work is organized in a way that it can happen independently, yieldeding benefit in terms of relieving some of the tension that collaborative development teams often experience. These principles of collaboration followed at the operational level are in line with and supported by the choice to use DVCS and GitHub, and influenced by the OSS culture and work organization.

In asking why these principles work, however, we should

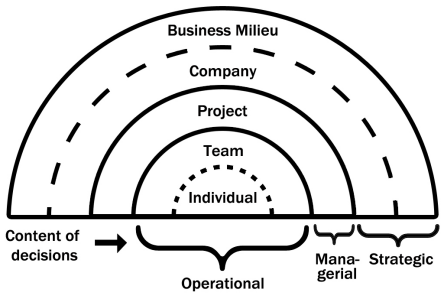

Figure 1: Decision layers in our analysis, from [12]

consider the following factors:

- **At the operational level the nature of the work allows independence and self-organization.** Some decisions that belong to the strategic and managerial layers benefit from multiple view points and brainstorming, such as the vision and identity of the organization, the business logic behind the software developed or the value that it will provide to the client. These decisions also involve expertise that is not part of the developers' skill set. Tasks at the operational level are highly modular and developers further break them into issues that are clearly defined, are entered in the GitHub issue tracker and can be picked up by members of the team.
- **At the operational level members are familiar with the idea of working this way and share the mentality behind it.** The principles of the code-centric perspective of collaboration are heavily influenced by the way work is organized and carried out in OSS projects, with which our interviewees have a lot of experience. As OSS developers themselves, they are self-driven and they share the mentality of self-organizing, minimizing communication and coordination needs, having ownership of code, and operating on a meritocratic, expertise-based model [16].

The cases of interviewees with a managerial role confirmed the benefits and the shifted perspective. These were, however, managers that are also programmers, using GitHub to contribute code as well as to maintain a holistic view of the project's status and progress for managing resources, and supporting the development team. The challenges happening in interaction with management, mentioned in Section 4.4, are cases of managers that are not GitHub users. However, the challenges may not be owing to the lack of GitHub use, but to the difference in mentality and perspective of collaboration that the two groups/levels have. On the operational level, programmers' sources of inspiration are DVCS, supporting decentralized work, and OSS, following practices of self-organization, environments that are usually not relatable to members of the managerial or strategic levels.

Studies have previously looked into the processes that have proven effective for OSS projects and how these could be implemented and benefit traditional organizations. Sharma et al. [30] examined OSS projects based on a framework of structure, process and culture. Their proposal was the creation of a hybrid-OSS environment in the organization that functions on technical professionalism, self-governance, community building, and trust. More recently, Riehle et al. [26] contrasted the OSS principles of meritocracy and self-organization to closed source software principles of rewarding status rather than merit, and imposing process. Their suggestion was that the software industry should capitalize on the OSS paradigm to bring benefits of internal open collaboration. They proposed channelling effort towards having motivated contributors, allowing them the room for volunteer action, and promoting better quality through quasi-public scrutiny.

Our study brings evidence on how the OSS paradigm of work is currently utilized in a commercial setting, and what are the potential benefits. Given that we arrived at this by observing the use of specific tools and the realization that these observations apply on one level of development work, below we pose some questions about the transferability and adoptability of our findings beyond the environment in which we have observed them.

5.4 Implications for future work

Building on our findings we have seen that at the operational level of development applying a code-centric perspective of collaboration, based on OSS principles of organizing and managing work, helps deal with some well-known collaborative difficulties. However, different parts of the development cycle or levels of decision making might have different communication and information requirements. We, therefore, use the understanding gained from discovering how teams on the operational level use GitHub, the OSS paradigm of work, and think about collaboration, to pose further questions.

a. **What are the factors that make this paradigm be used successfully?** We have identified some that are closely tied to development tools and environments, but more research is needed to see if there are others, by taking a more holistic view of development activities.

b. **Are the principles behind code-centric collaboration transferable to other levels?**

   i. **could an expertise-focused model that encourages independent work and self-organization, provide the same benefit there?**

   Decisions made on the managerial and strategic decision levels are further away from implementation and involve more creativity and abstraction. Such activities usually benefit from brainstorming and discussing ideas, which would seem to run against the principles of members working in isolation. The advantages and disadvantages of using this model for organizing work need to be investigated.

   ii. **does the application of those principles hinge on the use of specific tools?** Could (should) GitHub integrate with other artifact management systems used in other levels?

   Many of the interviewees reported challenges with getting non-technical members to adopt GitHub. Does that mean that if the tool cannot be adopted neither can the perspective of collaboration on which it operates? Is there room for tools used on other levels

to integrate with GitHub as a means of reaping some of the rewards experienced at the operational level?

iii. **how would managerial activities be affected if this model of work would be adopted on all levels?** What would be the potential trade-offs?

Central to the code-centric perspective of collaboration is the notion of team members focusing on tasks that utilize their expertise, while trusting others to do the same independently. The same can work for higher levels if management is willing to adopt it as a view point, applying the principles not only horizontally, within the same level, but also vertically, in interactions between levels. Horizontal application would mean that the members of the managerial and strategic levels focus on the tasks that utilize their expertise, while separating their concerns from other levels. Vertical application would mean that there is less need for monitoring and control activities. Management's role would be expected to shift towards supporting the operational level, by following a practice of trust, visibility, and correct recruitment.

iv. **Is the perspective driving the tool or vice versa?**

Certain technical capabilities enable different forms of organizing individual and collective work and, by extension, the collaboration of team members. In our case, the use of DVCS goes hand-in-hand with a high degree of independent work and a different view of collaboration. By the same principle however, it could be argued that the collaborative needs and views of teams change, opening the arena for the creation and improvement of tools, such as the imbibing of the OSS work mentality that we witnessed. While there is evidence of a relationship, it is difficult to pinpoint which is the driving force, since we see them evolving typically at the same time. An example of the same principle is seen when interpreting Conway's Law [8], where the presence of a homomorphic relationship between the system and the designing organization does not answer the question of which structure proceeded the other.

6. CONCLUSION

In this paper we have presented a study exploring the use of GitHub by collaborative development teams, in the commercial and OSS environment. We reported the mechanisms through which GitHub (building on DVCS capabilities) supports each of the collaboration elements, thus answering our research questions of how it supports collaboration overall. Our evidence (Table 3) shows that the major effects seem to be on minimizing and focusing the need for coordination, supporting GitHub's aim of promoting "collaboration without upfront coordination" [2].

Our work offers three contributions. First, we have seen how GitHub is used in commercial software projects and companies, by investigating activity that is beyond analyzing public GitHub events. Discovering how teams use GitHub to carry out their collaborative activities revealed their workflow and collaboration with less friction. The teams we have interviewed have formulated work practices around the tools and features available to them, utilizing the visibility of their collaborative environment to progress with their individual work, and minimize the explicit coordination effort by the adoption of pull requests as signals of upcoming synchronization points.

Second, through a systematic qualitative analysis of the interviews we identified the principles behind the benefits: *Focus*, *Independence*, and *Exposure*. The programmers' code-centric view of collaboration goes in a different direction than the more traditional view of collaboration as joint work but, nevertheless, showed clear signs of benefit associated with it. The inspiration for these principles of independent, self-organized work is strongly related to the OSS development environment mentality and the use of DVCS to manage a collaborative codebase.

Abstracting from the code-centric perspective, we took a stratified view of decision making in the development life cycle. The code-centric view is equivalent to a model of organizing work that is expertise-based and enables a separation of concerns between decision layers requiring different expertise. Since the adoption of a decentralized and OSS-like work organization model is proving successful and beneficial on the operational level of commercial development projects, we contribute to further research by posing questions to assess its applicability on other decision levels.

7. ACKNOWLEDGMENTS

We kindly thank all the participants of the study for taking the time to provide us with their valuable input and insight. Without them this study would not have been possible.

8. REFERENCES

[1] 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.

[2] F. Bolici, J. Howison, and K. Crowston. Coordination without discussion? socio-technical congruence and stigmergy in free and open source software projects. In *2nd International Workshop on Socio-Technical Congruence, ICSE*, Vancouver, Canada, 19 May 2009.

[3] 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.

[4] 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.

[5] 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.

[6] M. Cataldo, P. A. Wagstrom, J. D. Herbsleb, and K. M. Carley. Identification of coordination requirements: implications for the design of collaboration and awareness tools. In *CSCW '06: Proceedings of the 2006 20th anniversary conference*

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

on Computer supported cooperative work, New York, NY, USA, 2006. ACM.

[7] S. Chacon. *Pro Git*. Apress, Berkely, CA, USA, 1st edition, 2009.

[8] M. Conway. How do committees invent. *Datamation*, 14(4):28–31, 1968.

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

[10] 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.

[11] 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.

[12] B. Curtis, H. Krasner, and N. Iscoe. A field study of the software design process for large systems. *Commun. ACM*, 31(11):1268–1287, Nov. 1988.

[13] 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.

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

[15] S. Faraj and L. Sproull. Coordinating Expertise in Software Development Teams. *Management Science*, 46(12):1554–1568, 2000.

[16] 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.

[17] G. Gousios, M. Pinzger, and A. van Deursen. An exploration of the pull-based software development model. In *ICSE '14: Proc. of the 36th Int. Conf. on Software Engineering*, June 2014. To appear.

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

[21] 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.

[22] 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.

[23] B. O'Sullivan. Making sense of revision-control systems. *Commun. ACM*, 52(9):56–62, Sept. 2009.

[24] 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.

[25] D. Redmiles, A. Van Der Hoek, B. Al-Ani, T. Hildebrand, S. Quirk, A. Sarma, R. S. Silva Filho, C. de Souza, and E. Trainer. Continuous Coordination. A New Paradigm to Support Globally Distributed Software Development Projects. *Wirtschaftsinformatik*, 49:28–38, 2007.

[26] 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.

[27] 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.

[28] A. Sarma, D. Redmiles, and A. van der Hoek. Categorizing the spectrum of coordination technology. *Computer*, 43(6):61–67, 2010.

[29] 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.

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

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

[32] 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.

[33] 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.

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

[35] 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.

[36] 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.