# MODELING DISTRIBUTED COLLABORATION ON GITHUB

In this paper we apply concepts from Distributed Leadership, a theory suggesting that leadership is shared among members of an organization, to frame models of contribution that we uncover in five relatively successful OSS projects hosted on GitHub. In this qualitative, comparative case study, we show how these projects make use of GitHub features such as pull requests. We find that projects in which member pull requests are more frequently merged with the codebase experience more sustained participation. We also find that projects with higher success rates among contributors and higher contributor retention tend to have more distributed (non-centralized) practices for reviewing and processing pull requests. The relationships between organizational form and GitHub practices are enabled and made visible as a result of GitHub's novel interface. Our results demonstrate specific dimensions along which these projects differ and explicates a framework that warrants testing in future studies of open source software, particularly GitHub.

*Keywords*: GitHub, Distributed Leadership, Open Source Software

## 1. Introduction

Medium to large-scale open source software (OSS) products rely on volunteer contributions, and have long been a staple environment for investigation by researchers interested in open collaboration and virtual organizations. A substantial literature exists, for example, focused on the examination of SourceForge, an open source repository that started in 1999. More recently implemented OSS repositories include social features that bring a new transparency to the development process [4]. GitHub, for example, is a collaborative, web-based code-hosting service built on top of the Git version control system. GitHub is popular [25], with 3 million users and nearly 5 million hosted repositories as of January 2013 [18]. Greater understanding of how distributed collaboration on GitHub occurs will provide a foundation for modeling, understanding and emulating successful patterns.

Distributed collaboration is a foundational characteristic of OSS software, but a characteristic with many variations in practice. Prior studies, for example, frame OSS projects as meritocracies that encourage multiple people to take on leadership roles over a short- or long-term period [21]. Some OSS projects, like Linus Torvalds' Linux, are described as being led by a "benevolent dictator". On such projects leadership is clear and explicit. Comparisons across projects with different leadership styles, however, are rare. Yet such comparisons are critical for project founders looking to discern successful approaches from the experiences of others.

One critical gap in the OSS literature is that notions of flexible and informal leadership, in which more than just one or two developers are responsible for leading, remain unexplored. Another gap emerges around the examination of practices and participation models associated with successful projects. The comparison of projects on GitHub, therefore, fills an important gap in our understanding of distributed collaboration in OSS.

Open source projects tend to attract a range of individuals and incentive models: e.g. the pure (unpaid) volunteer, the resume builder, the industry-backed contributor or the paid contributor. When contributors first join an OSS project, other members review their code contributions before they are incorporated into the product. Over time, these

new contributors can take on increased responsibility and contribute directly to the codebase.

GitHub's "fork & pull" model allows developers to contribute to a project without being granted permissions, thus removing a critical barrier required by prior OSS models**.** In this model, users "fork" the repository to create their own personal copy of the source code. Users then make changes to their copy and submit a pull request (PR) to alert the project maintainer that they would like their changes to be pulled into the main branch of code. PRs are often used to initiate a code review or discussion around commits. GitHub also allows users to be granted push/pull access to a repository, giving them the status of 'project collaborators' who can commit their code changes directly to the main branch and merge pull requests into the main branch.

The distributed nature of GitHub's technical design, combined with the social practices emerging around it, would seem to naturally support projects with less centralized leadership models. In less centralized organizational models, leadership is conceptualized as a shared attribute of the organization [9]. Instead of focusing on leadership as a characteristic of individuals within a project, distributed leadership (DL) models operationalize collaborative control through the participation of members. In this paper, we focus on contribution growth and access to privileges within a GitHub repository as key components for understanding how distributed leadership is enacted through practices on GitHub. Specifically, how projects on GitHub manage the acceptance of pull requests and the granting of merge access are key signals of distributed leadership and collaboration. Distributed leadership is a model that recognizes notions of spontaneous collaboration, intuitive working relations and new forms of institutionalized practice [9] that have emerged as dimensions of GitHub's pull request architecture [4].

Operationalizing leadership through DL theory provides a lens for understanding distributed, online collaboration in new ways. In particular, the role that technology plays [12] in the operationalization of more distributed forms of collaboration is an important thread in this study. Here, we build on the approach of Luther, Fiesler and Bruckman [12] in our application of DL to online collaboration. DL theory conceptualizes leadership as a socially constructed dynamic, and acknowledges that changes to leadership (or different styles or structures of leadership) may be taking place in a socio-technical environment like GitHub.

In the next section, we elaborate on DL as a framework for understanding how collaborator access and other project privileges are utilized on GitHub. We then present a qualitative, comparative case study of five popular OSS projects on GitHub. Within each case study we identify several leadership indicators by reflexively analyzing trace data mined from GitHub using GitMiner (https://GitHub.com/pridkett/GitMiner) as well as qualitative interview data. This paper contributes the identification 5 factors for comparing distributed leadership in GitHub communities, as well as an elaboration of a method that combines qualitative interviews with the analysis of trace data from GitHub. We explicate each category and conclude with an exploration of the implications for DL theory development in virtual organization contexts, as well as models of success in this new type of OSS project environment.

## 2. Literature Review

Prior research explores models for OSS development that consider OSS organizations a "pyramid meritocracy" [6,10,20]. A pyramid meritocracy is hierarchical and centralized but has no central authority. Instead, leadership may be shared among a group of core developers who "act as peers at the top echelon of the pyramid" [20]. This pyramid consists of "regulars" who form the lower level, emerging leaders, who form the middle, and leaders, who constitute the topmost level. Another characteristic of pyramid meritocracy is that it tends to favor gradual innovations over radical ones. While radical system changes may be embraced by some members of the group, such changes often result in the creation and maintenance of a separate branch of the software; a permanent, dissonant software fork, which is distinct from GitHub's implementation of forking as a principle step in distributed collaboration [20] (On GitHub, the fork is considered temporary). Others conceptualize projects with layers around multiple centers [5,20]; several empirical studies of these types of projects find that a small group is often responsible for most of the work and a large group of peripheral participants are responsible for the rest [8,13,16].

Low barriers to participation on GitHub, such as ease of forking and the ability to submit pull requests, may cause a shift in this structured "pyramid" meritocracy towards more distributed meritocracy. How, and to what extent the meritocracy conventionally found in open source software is influenced or fundamentally changed through the use of GitHub is not well understood.

Dabbish, Stuart, Tsay and Herbsleb [4] demonstrate the value of transparency on GitHub: Members make a "surprisingly rich set of social inferences from the networked activity information" and "combine these inferences into effective strategies for coordinating work, advancing technical skills and managing their reputation." [4]. Marlow, Dabbish and Herbsleb [15] describe impression formation on GitHub, focusing specifically on how social activity streams enhance member receptivity to contributions through pull requests. It is possible that further removing frictions in coordination may also mitigate the need for formal leadership in terms of organizing members towards a common goal. Moreover, the fact that contributing to one repository is like contributing to them all from a technical standpoint means that less social influence is required to formulate and disseminate rules (bureaucracy).

Ducheneaut [5] offers a picture of OSS organizations founded on studies that aggregate statistics about OSS projects based on message postings [17,24], commits and downloads [7] and role structures [14,17]. Ducheneaut depicts the pattern of organization as "a series of concentric circles" [5]. The center circle contains the core developers; surrounding the core is a ring of "maintainers" responsible for one or more modules of a project. Extending farther from the center are rings of "bug reporters", "patchers" who fix bugs, "documenters" and "users." As Ducheneaut [5] points out, even users can be "highly skilled"; thus, this periphery involves a "nebulous arrangement of participants." Whether or not this view of OSS project teams as role hierarchies remains a salient theory for explaining OSS organizations, we know very little about how members of a particular community exhibit leadership behaviors at various times.

Von Krogh, Spaeth and Lakhani's [11] study of Freenet examines how "joiners" become "newcomers", which they define as members with access to directly change code. OSS projects, then, are shown to exhibit a number of different structures in separate

studies. A qualitative, comparative case study of multiple OSS projects in a single socio-technical context, like GitHub will help to frame a reconsideration of OSS leadership. There is no one model of OSS leadership on GitHub, any more than there was one model in studies of SourceForge. The present study fills a gap in describing the range, and styles of leadership found on GitHub, focusing especially on the nature of distributed leadership and how it emerges in a more *technologically* distributed source control system.

### 2.1. Distributed Leadership

Recent literature questions leadership theory that focuses on leadership as a concept in which "knowledge and direction trickle down from a notional 'top,'" and which portrays leaders as charismatic individuals who proactively manage pyramids of people [22]. Distributed Leadership is one specific, more contemporary theory that argues that leadership can be a "shared attribute" of an organization. Through this theoretical lens, members of a community are seen as leaders through their actions and interactions within the socio-technical context of OSS software development in general, and GitHub in particular.

To explore leadership through the DL lens, Gronn argues for a shift in the unit of analysis from an aggregation of individuals towards viewing leadership as distributed, concertive action. "Rather than aggregated, individual acts" we should be examining concertive action in three patterns: spontaneous collaboration, intuitive working relations and institutionalized practices [9]. We elaborate on these conceptual cornerstones of distributed leadership in the following sections.

#### *2.1.1. Spontaneous Collaboration*

Spontaneous collaboration can arise "in response to particular problems and requirements" [23]. Gronn elaborates further:

> *"One way is when sets of two or three individuals with differing skills and abilities, perhaps from across different organizational levels, pool their expertise and regularize their conduct to solve a problem, after which they may disband. These occasions provide opportunities for brief bursts of synergy which may be the extent of the engagement or the trigger for ongoing collaboration." [9]*

Thus, spontaneous collaboration emerges in response to problems or requirements allowing groups of individuals with disparate skills or responsibilities to solve a problem. After a problem is solved members may disperse, but the experience of collaboration creates an opportunity for future engagement and collaboration.

#### *2.1.2. Intuitive working relations*

Intuitive working relations emerge through practice between at least two people who share interdependencies, which results in a "shared role space" [9]. Gronn elaborates on this concept:

> *"In the second instance, intuitive understandings are known to emerge over time when two or more organization members rely on each other and develop a close working relationship. In this instance, leadership is manifest in the shared role*

*space encompassed by their partnership. It is the working partnership as a focal unit which is attributed with leadership by colleagues, and the partners are aware of themselves as co-leaders." [9]*

#### *2.1.3. Institutionalized practice*

Institutionalized practice is "seen in the tendency to institutionalize formal structures"; these practices are "formalized either by design or by adaptation" [9]. Gronn posits that an example of this might be a "temporary task force" which may later be "incorporated into an organization's formal framework of governance" [9].

Scacchi's concept of *Software informalisms* [19] connect Gronn's notion of institutionalized practice to the context of GitHub. Software informalisms "are the information resources and artifacts that participants use to describe, proscribe, or prescribe what's happening in a OSS project. They are informal narrative resources that are comparatively easy to use, and publicly accessible to those who want to join the project, or just browse around" [21]. Common formats that informalism take are email lists, discussion threads, group blogs, news postings, internet relay chat (IRC), use scenarios on the projects web pages, how-to guides, project wikis, as well as system documentation and publications [21]. Software informalisms can replace formal "requirement specifications" and can capture the rationalizations and debates surrounding the reasons that changes to a particular piece of code are made, as well as other development activities and learning artifacts [21]. That is, institutionalized practices may never be "formalized," but are rather the by-products of coordination and socialization. "Simply put, by doing things, we create the way to do things" [3].

## 3. Operationalizing Distributed Leadership on Five GitHub Projects Key Terms

Gronn's three patterns of concertive action frame our analysis of DL in OSS on GitHub. The application of distributed leadership to the study of OSS is novel, as our review of the literature reviews only one prior study that applied the concept of DL to online collaborations to "account for the roles of people and technology" [12]. In the next section we discuss specifically how our study of GitHub is framed using DL. Ultimately, we define three attributes that present possible measures for grouping communities in terms of their use of the technology to distribute responsibility in the context of collaborations.

Leadership is a complex concept. For the purposes of our research, we limit our understanding of leadership to encompass those practices that lead to shared community involvement and responsibility for technology – those characterized by our definitions of the components of DL (i.e. spontaneous collaboration, intuitive working relationships, and institutionalized practice).

Before we operationalize concepts of DL in OSS, it is necessary that we define some key terms:

**Commit** – A change to a file or set of files.

**Collaborator** – A GitHub user with push/pull access to a repository.

**Fork** –A personal copy of another user's repository where a user may make changes in isolation from the original project repository.

**Pull Requests** –Proposed changes to a repository that may be accepted or rejected by a project collaborator.

**Issues –**Documented bugs, problems, feature requests or questions associated with a repository.

**Maintainers** – GitHub users who have administrative privileges to a repository, and thus the privilege to grant members access to commit.

**New Contributors** – GitHub users submitting a pull request to a project for the first time.

### 3.1. Spontaneous collaboration – Receptivity to New Contributors

Gronn described spontaneous collaboration as people coming together in an emergent fashion to solve a problem or respond to a new requirement. Collaborators may or may not ever work together again. To operationalize this dimension of distributed leadership on GitHub, we emphasize a) the problem *solving* focus of Gronn and b) the impermanence of such collaborations. A key indicator of an accepted solution on GitHub is the actual acceptance of a pull request. Though many pull requests result in fruitful discourse that may lead to improvements in a project (or not), the accepted pull request is an indicator that a problem was *solved*. Therefore, we operationalize "spontaneous collaboration" as each time a new contributor submits a pull request that is accepted and merged into the main branch of code.

Episodes of spontaneous collaboration are indicators of distributed leadership in Gronn's theory. Therefore, we regard communities that have higher merge rates as having more distributed leadership. The extent to which projects are receptive to new contributions (i.e. merging contributions from outsiders) is a measure of their acceptance of spontaneous contribution. Communications around merged pull requests are indications of a distributed group working together. The actors and the surrounding code serve to indicate any acts of collaboration taking place before or as evidenced by their communications in the pull request.

Our focus is on the utility of emergent collaborations and how and to what extent they shape community practice. Moreover, how does the timing and success of these collaborations affect outcomes? To answer these questions, we look at the rate of PR acceptance of active contributors on five projects using GitHub. We define 'active' as members with open pull requests at the time these data were collected.

### 3.2. Intuitive working collaborations – Gaining Access

Pull requests are often the first, most distant way that outsiders to a project initiate new contributions on GitHub. Among contributors, some subset will become more central collaborators on a GitHub project. The extent to which this is more common, a project is operating in a distributed leadership model.

We operationalize institutive working relations as emerging from more distant collaboration, and ultimately as measurable through the incidence of individuals gaining the ability to commit directly to the main branch; to become collaborators. In this operationalization of the intuitive working relations dimension of distributed leadership, repositories on GitHub with larger numbers of members with collaborator access have greater distributed leadership.

Individual project teams decide who is granted collaborator access either informally or through some formal voting structure [6]. Commit responsibility typically comes with the responsibility of reviewing contributors' PRs. We look to identify how extensively communities rely on the distributed work of a group of members with access to merge code that is submitted through PRs.

### 3.3. Institutionalized practice – Merge Behaviors and PR Review

For Gronn, the unit of analysis is "distributed practice" rather than individuals; that is, he distinguishes leadership roles from leadership practice or behaviors and takes up the latter as the unit of analysis. DL centers on the contextualized outcome of interactions among "geographically dispersed couples, co-leaders and partners, triadic role constellations, and rotating leader systems" [9] within a community or organization. DL "allows for the possibility that all organization members may be leaders at some stage" [9]. Turning to OSS, we might ask how leadership is distributed among "members" of the community with access and "contributors" to the community, both of whom submit code and comment on code. While there may be several ways that institutionalized practice is manifest, we consider that communities where mergers (members with access) subject their pull requests to community review are more distributed than those where members with access bypass this process. We also consider that those communities where responsibility for merging others' code is more distributed exhibit greater distributed leadership. Gronn defines institutionalized practice as the introduction of formal structure and process. Critically, we look for indicators that distributed leadership markers like spontaneous collaboration and intuitive working relations are institutionalized.

GitHub projects use different approaches for managing commits, particularly among those members with commit access. One group of projects compels even members with commit access to use the pull request process, except in the case of minor code changes. The institutionalizes transparent review of their code by other project members in order to implement major code changes**.** Another group of projects in which PR review is enforced among members with commit access are shaped differently than those in which PR review is not required for members with commit access. In these cases, the rules apply, except to the group of people who are in roles with some leadership responsibility. A third group of projects us pull requests in all cases, including the repository owner's changes. This practice makes pull requests an engaging and practical review process, as well as a symbolic gesture of open leadership enabled by GitHub's design. Such practices make major changes open to the community and suggest that greater consensus is necessary even for changes made by the maintainer and core members – which reflects a more open and democratic approach than is reported in studies of OSS where traditional, centralized models of leadership are more salient.

These three configurations of concertive action are described by Gronn as "conjoint agency"; where "agents synchronize their actions by having regard for their

own plans, those of their peers, and their sense of unit membership" [9]. Conjoint agents are not purposely acting upon common interests, though their interests may be advanced through a coordinated effort, nor does their collaboration involve the relinquishing of some rights in exchange for the benefits of authority [9]. Gronn suggests that conjoint agency occurs more frequently due to networked computing.

The complexities associated with "fragmented and dispersed knowledge" have led to "alternative modes of articulating the flow of work, in particular, the redefinition and reintegration of tasks" [9]. In DL, contributors' responsibilities will necessarily overlap and may be complementary. Gronn refers to this as "interdependence" [9]. While the latter component, complementarity, may mean that members leverage their individual strengths in support of a common goal, it is seen as advantageous when members who rely on others "enhance their lesser skills through frequent shared talk and observation of each other" [9].

# 4. Research Goals

Based on the described framework, we aim to characterize communities on GitHub with the goal of learning from the different forms of organization. To do that, we must develop ways of describing project organization. In this paper, we provide three measures that we believe can be useful in reliably distinguishing among project organizations, using distributed leadership theory as the primary lens. We advance this argument by drawing on interviews conducted with members of these communities and existing leadership theory to drive the development of measures that utilize electronic, archival data. By undertaking a qualitative, comparative case study of five projects, we seek to explicate an approach for making systematic project comparisons on GitHub, and advancing understanding of the organizational forms that exist in that context. Ultimately, these measures should function to describe a larger set of projects on GitHub and inform future research. This paper focuses on answering two core research questions:

> RQ1 – How and to what extent are differences in distributed leadership described by project contributors on GitHub manifest in the utilization of Pull Requests?
>
> RQ2 – How and to what extent are the 3 specific components of distributed leadership manifest across the five different GitHub projects examined?

# 5. Data Collection and Methods

### 5.1. Project context

We investigated five popular projects on GitHub (Table 1). We operationalize project popularity by the number of forks per project. Before GitHub, the term 'forking' had a negative implication in software development, denoting developers who split from the project community to take the code in a new direction. However, on GitHub, active projects often have many forks, as this is the primary way contributors make changes to the codebase that they hope to merge with the main project. The number of forks, therefore, is an indicator of project popularity. We selected projects to study based on a diversity of project popularity, as indicated by fork count, and project purpose, as

indicated by a description of the project. All five projects seek to create various types of software developer or software infrastructure tools.

| Project | Project Organization | Dates of data collection |
|---|---|---|
| **JavaScript Library (JS)** | Volunteer & paid developers with support from non-profit Foundation | 2010-09-02 – 2012-12-12 |
| **Web-Application Framework (WFc)** | Volunteer & paid developers with support from non-profit Foundation | 2010-09-05 – 2012-12-10 |
| **Web-Application Framework (WFr)** | Volunteer & paid developers with sponsorship by a web application company | 2010-09-02 – 2012-12-01 |
| **A Server Side Software System (SS)** | Volunteer & paid developers with sponsorship from a software and services company | 2010-08-31 – 2012-11-30 |
| **Front-end framework (FF)** | Primarily paid developers | 2011-08-19 – 2012-12-08 |

Table 1. Classification, description and dates of collection of each of five GitHub projects

Each of the projects we investigated operates slightly differently. The JavaScript library (JS) community is, perhaps, the most rigorously organized and invites a variety of contributions (including html for its website). The Web-application framework (WFc) is the smaller of the two Web-application frameworks. Their community depends primarily on volunteers who have adopted an open, "democratic" (in their own words) approach to leadership. The JavaScript Library JS makes a similar claim to being open. However, JS does have somewhat more rigorous protocols in place that differentiates them from all other communities; primarily, in their main software repository contributors are required to first submit an issue before making a pull request addressing that issue. This is somewhat of a bureaucratic measure.

The larger Web-application framework (WFr) has a reputation for being difficult to contribute to, however, their numbers suggest otherwise. The Server Side Software System (SS) says that the introduction of GitHub has lowered the coordination costs for software development drastically (SS2). Finally, the Front-end framework (FF) is, at first glance, seemingly the most hierarchical. As the FF project has grown, the number of members besides the maintainers with access to commit (and thus, merge) has only grown by one.

### 5.2. Qualitative Data Collection

We conducted semi-structured interviews with 16 maintainer and core members (Table 2) from four of the five large GitHub projects studied as part of this research. Contributors to each project were recruited via email and were given no incentives. Each project has had over a hundred contributors. Two projects, the JavaScript Library and the Web-application framework, have switched to GitHub from a different code hosting infrastructure. Interviews lasted approximately 45 minutes and were conducted over Google Hangouts and Skype (and one via email). In addition to asking about informants' current and past project roles and how their responsibilities for work had changed, we

focused on how new members gain commit access; pull request protocols; how the project had evolved on the platform, how the respondent and their community use its features and coordinate work and how they measure "success" of the project. Interviews were recorded, transcribed and coded based on emergent themes. We draw on our findings from these interviews throughout the paper.

| Project | Participants |
|---|---|
| JS | JS1, JS2*, JS3, JS4, JS5, JS6 |
| WFc | WFc1*, WFc2 |
| WFr | WFr1, WFr2, WFr3, WFr4* |
| SS | SS1*, SS2 |
| FF | (declined interviews) |

Table 2. Interview Sample (*Project Maintainer)

### 5.3. Integrated, Reflexive Data Analysis

For each project, we collected data on pull requests, issues and commits including all associated comments and details on GitHub users who interacted with these items. The data was collected starting when each project first began using GitHub – which is 2010 for all repositories except FF (see Table 1 for exact dates of collection). Further, we interviewed participants and leaders in four of the five projects to triangulate our mined data with participant experiences and perceptions (one project declined to be interviewed).

To collect the data, we used GitMiner (available on GitHub: https://GitHub.com/pridkett/GitMiner), a Java-based tool for extracting data from GitHub to a graph database. GitMiner downloads project data using the GitHub API and interfaces directly with Git to obtain commit information. To run GitMiner, a configuration file is used to specify which projects should be downloaded and what type of information should be gathered for each project. Table 1 describes our selected projects, their stated organization properties, and dates of data collection.

We analyzed the interaction data mined using GitMiner and the interview data reflexively, following the systematic approach of Group Informatics [2]. Our comparative analysis of interview transcripts guided our connection of Gronn's theory of DL with specific aggregations of GitMiner data: (1) spontaneous collaboration is examined as receptivity to new contributors, (2) intuitive working relationships are examined as gaining access, and (3) institutionalized practice is examined as PR review and merge behaviors. Our findings build understanding about how to provide measures which group projects in the spectrum of distributed leadership. (For summary, see Table 3.) We argue that future studies will be able to use this framework as a guide for building theories of virtual organization that are salient to GitHub, and in other contexts where DL is a central feature.

| Potential indicators: | **Gronn's Distributed Leadership** |
|---|---|
| **Contribution Success: What percent of Contributions are Merged?** | Spontaneous Collaborations |
| **Is access to merge distributed?** | Intuitive working relationships |
| **Do they adopt distributed PR Review practices?** | Institutionalized practice |
| **How distributed is responsibility for merging other people's code?** | Institutionalized practice |

Table 3. Operationalizing DL in OSS Communities on GitHub

# 6. Findings

Our operationalization of Gronn's three concepts of DL discerns specific contrasts among the five cases, leading us to suggest that three models of leadership are visible across the cases we examine – Distributed Collaboration (WFr, WFc and JS), Distributed Coordination (SS) and Highly Centralized (FF). The answers to our two research questions characterize the differences in these five communities more fully.

## 6.1. How and to what extent are differences in distributed leadership described by project contributors on GitHub manifest in the utilization of Pull Requests?

In our interviews, informants consistently reference pull requests as the central feature of GitHub around which project leadership takes place. Our informants consistently point to the transparent, public nature of GitHub as elements that enable participation. GitHub users experiment with and adopt the features of a decentralized source code management system; most notably, pull requests are the dominant mechanism used to contribute code and are one center of interaction.

Our informants are clear and specific in their comparisons of GitHub to platforms they have used previously, and describe GitHub's interface and tools as a significant factor contributing to participation in their projects. Specifically, they note that the process of submitting a patch is common knowledge, and this has lowered barriers to contribution on GitHub (WFc1, JS2, WFc2, JS2, SS1, SS2). Informants also say that more people seem to be contributing to OSS projects on GitHub through the mechanism of pull requests (P1, P3, P6, P11, SS2). Developers note that user experience on GitHub allows projects to operate in more "democratic" (WFc1, WFc2, JS6, SS2) and "transparent" (WFc1) ways. This transparency leads to greater participation and to opportunities for review and feedback from project members (JS2).

The barriers to entry experienced by new contributors to GitHub projects are low compared to those of other OSS tools that our informants have used (WFc1, WFc2, JS2, WFr1, WFr2, WFr3, WFr4, SS1, SS2), increasing the number of GitHub projects our informants participate in. One respondent (P6) notes that other platforms require a great amount of effort to publish software, whereas changes on GitHub can be published and small projects initiated with little effort. For these reasons, GitHub is "accessible to a class of people that it wasn't before" (P6).

Gathering an understanding of how a project works and where a project team is currently focused is one specific barrier our informants report in prior OSS code-hosting experiences (mostly SourceForge) (WFc2, WFr2). GitHub provides easy access to revision and comment history (WFc2, WFr2), overcoming the opaqueness of OSS projects hosted elsewhere (JS1). In addition, the ability to quickly create a replica of any project ("forking") and begin working with its code is another specific advantage of GitHub (WFc1), as compared to other tools. In practice, "forking" prevents users from needing to reconstruct changes they make over weeks or months due to changes in the

main code repository. On GitHub, the user synchronizes their fork with the main project, then submits their changes as a pull request.

GitHub projects are perceived by our informants as more consistently managed; and therefore, they find it easier to apply the practices used on one project to make contributions to another (JS1, WFc1, SS2). “There's no mystery on how to support a patch" (WFc1), for example. Consistency across projects means that users do not have to "learn how it works" because they all "kind of work the same" (WFc1). This “common language of contribution” infers that new participants and new projects on GitHub do not need to dedicate as much time to finding the best way to contribute. This contrasts sharply with our respondent experiences on other OSS platforms, where “every [project] had [its] own way of doing things” (SS2).

There is a strong sense among our informants that the level of participation on GitHub is much higher than on other OSS projects that they have previously contributed to outside of GitHub (WFc1, WFc2, SS2, WFr1). One project that moved from an older OSS repository to GitHub noted that the level of participation has “doubled” since moving to GitHub (WFc1, WFc2):

> *“[Moving to GitHub] was probably one of the best moves we ever did for the project. The number of contributions has gone up from – it was in the low twenties; it’s over 150 now people who have contributed, so it’s much better.” (WFc1)*

It is evident from our interviews with informants, and analysis of the traces of pull requests (covered further in RQ2) that discerning categories of projects on GitHub centers on the specific utilization of pull requests by particular projects. While there are other technical dimensions of GitHub that facilitate interaction, our informants are consistent in referencing pull requests as the central feature for understanding the implementation of leadership, and the distributed nature of leadership, on particular projects.

### 6.2. How and to what extent are the 3 specific components of distributed leadership manifest across the five different GitHub projects examined?

#### *6.2.1. Spontaneous Collaboration and Contributor Merge Success*

We operationalize Gronn’s DL pattern of spontaneous collaboration as contributor merge success rate. Our interviews verify that pull requests are being used to monitor redundant contributions, and facilitate collaboration where multiple contributors have identified the same, specific desired change.. In Figure 1 we show the total number of contributors with at least one merged pull request. Table 4 shows the success rate for any given number of PRs contributor. For example, in community SS, 14 members have submitted single pull requests with an overall success rate of 78%. JS has the highest receptivity to contributions, with 64% of all members contributing code that was successfully merged, followed by WFr (58%) and WFc (46%). However, projects with high receptivity to contributions are not always those with the highest contributor volume. Our two lowest volume communities (WFc and JS) and our highest (WFr) have percentage of PRs that are merged at or above 50%, whereas our two moderate to large contributor volume communities have dramatically lower merge rates (SS at 3% and FF at 14%). We also find that communities with higher merge rates among contributors are also those with members who continue to contribute. That is, projects with more spontaneous

collaboration (higher rate of merges) have greater contributor retention – i.e. more members who have made 4 or more contributions.

Table 4 shows that communities with low merge rates (see Table 4, Figure 1) have dramatically low numbers of repeat contributors compared to communities with high merge rates. In communities SS and FF, which have only a small percent of contributors with successfully merged pull requests, only one in five people have more than one merged pull request. In WFc and WFr, where around half of contributors have had a PR merged, around two in five have had more than one pull request merged. In JS, which has the highest merge rate for contributors (64%), the percent of people contributing successfully more than once is around 30%.

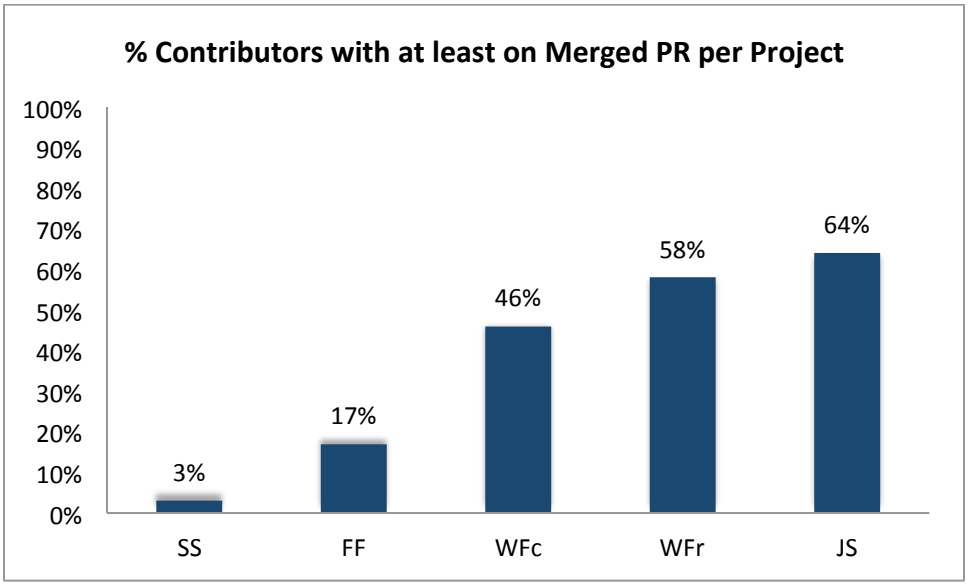


Fig.1. the percentage of contributors with at least one merged change per project

| | SS | | FF | | WFc | | WFr | | JS | |
|---|---|---|---|---|---|---|---|---|---|---|
| **PR Success Rate** | % | # | % | # | % | # | % | # | % | # |
| **Contributors who have contributed … 1 PR** | 78% | 14 | 82% | 142 | 55% | 60 | 59% | 445 | 71% | 229 |
| **2 PRs** | 5.5% | 1 | 8% | 14 | 15% | 16 | 17% | 126 | 15% | 47 |
| **3 PRs** | 0% | 0 | 5% | 8 | 6% | 7 | 7% | 55 | 4% | 13 |
| **4 to 10 PRs** | 11% | 2 | 5% | 9 | 16% | 18 | 12% | 88 | 6% | 19 |
| **> 10 to 20 PRs** | 5.5% | 1 | 0% | 0 | 4% | 4 | 2% | 18 | 2% | 5 |
| **> 20 (148 max) PRs** | 0% | 0 | 0% | 0 | 4% | 5 | 3% | 21 | 3% | 10 |
| | 100% | 18 | 100% | 173 | 100% | 110 | 100% | 753 | 100% | 323 |

Table 4. Contribution retention: percent/number of active contributors who have submitted at least one PR that was merged by contributor volume

Our interviews confirm that retaining contributors is clearly of concern among communities with high merge rates. Specifically, JS is clear about the need to be "kind" when addressing new participants, particularly because many have had the experience

where their first submission doesn't 'land.' That is, they perceive that people walk away when they are not successful on their first try or that they are offended when they don't receive acknowledgement for their effort. JS has adopted "boilerplate" responses that include "pleases" and "thank you's" with instructions about what to do so as not to discourage new participants and also guide them through what they must do in order to make their submissions acceptable. Practicing diplomacy, they say, can be as simple as saying "thank you for your contribution." There is a designated role in JS for those who identify new talent and make sure that they feel welcome. This role used to be referred to as "evangelist" (JS6).

> *"We need to be sure that we had better boilerplate [responses] when people contributed bedbugs on the bug tracker or when someone comes along with their first contribution making sure that whoever is looking at it know[s] it's their first contribution and that they should cut them a little bit of slack, instead of just you know – that typical sort of open source you should have known it already attitude … you take someone across the precipice, I guess. (JS6)*

> *"The very first thing to do is be kind … For example I triage bugs and when someone will make a test case and they'll spend their time writing it and you close it [and say], "Sorry, won't fix," … if you just say that … it offends them…. You just – in simplest writing –[say,] "thanks for contributing."(JS4)*

#### *6.2.2. Intuitive Working Relationships: Distribution of Collaborator Access*

We operationalize Gronn's intuitive working relationships pattern as distribution of access to merge privileges. We learn that commit access, or the ability to commit directly to the codebase without submitting a pull request, is typically granted to new contributors only after they have made substantial submissions (WFc2, SS1, WFc13). While movement from the periphery to active participation can sometimes occur very quickly (WFc2, WFc13) we are told that during one project most contributors are not invited to the **core** until they have been contributing for 1 to 2 years (WFr4). We learn through our qualitative interview with a contributor that on one project a "loose voting" process is in place, whereby core team members will elect certain contributors via email for collaborator access (WFr1).

People who make a sizeable contribution generally will get access to merge directly to the main branch if they're "interested" (WFc1). Notably, virtually all those with access who are now either members of the core or maintainers that we spoke with were contributors who had at one point submitted bug fixes, documentation features etc.

> *"But yeah, if you do enough submissions that are of – that everyone is current on the team and thinks is good, then we just give you access." (WFc1)*

SS suggests that receiving access is about knowing some portion of the code and demonstrating some enthusiasm and ability for reviewing others code and working with people in the community:

> *"I would say it's a combination of showing that they really understand at least some relevant portion of [SS], the project, and also that they have a good enough*

*– they kind of have demonstrated a good personality and ability to kind of work with people and get patches to the point where they're ready." (SS1)*

FF declined to participate in interviews, and we therefore do not know the motivations of their restrictive access practices. Our quantitative findings show that distribution of responsibility for merging is low, as only two members of FF are merging virtually all pull requests and one other person has merged a single pull request since the beginning of the project. Other members are making commits directly to the main branch, however. The inclusion of FF as a project is important for understanding the diversity of GitHub projects overall. FF, in its current state, has not merged any pull requests from non-leaders, which puts it in a category of projects with highly centralized, non-distributed leadership. The socio-technical infrastructure of GitHub enables the study of project practices, which gives us useful information even if we do not have access to their reflections about that practice.

#### *6.2.3. Institutionalized Practice: PR Review*

Our reflexive analysis of interviews and GitHub trace data focused our operationalization of Gronn's third pattern of DL, institutionalized practice, toward building understanding of the nuance in how pull requests are reviewed across projects. Responsibility for merging is most distributed among communities with high merge rates. While many people have responsibility for merging in community SS (which has low merge rates), these represent an extremely modest number of PRs (averaging 2.25) almost all of which are their own (see Figure 1). Essentially only two people are currently responsible for merging PRs on FF (which also has low merge rates) but all of these are contributed from outside, non-merging contributors; *FF members with merge access do not submit PRs*. We find that with the exception of FF, virtually all mergers have made or currently make PRs (see Figure 1). The average number of PRs merged by members varies substantially across communities.

Our two low merge rate communities (FF and SS) have two different merge profiles, while our three high merge rate communities are fairly similar. Among low mergers, SS mergers do submit PRs but mostly merge their own PRs. In FF, there are only two primary mergers who do not submit PRs and thus are exclusively merging other people's contributions.

Among high merge rate communities, mergers submit PRs and merge mostly other people's PRs. That is, while they review and merge others code, responsibility for committing their own code can and often does go to someone else. Table 6.1 – 6.5 (which can be found in the Appendix) summarizes these main differences.

We also validated our understanding that virtually everyone (besides the FF community), regardless of whether members have access, will submit PRs for major changes, features and other, less straightforward reasons (WFc1, P2, WFc2, SS1, JS3, JS4, JS5). This allows those who are unsure of the merits of their pull request to seek feedback from the community by submitting new code for review, resulting in code that 'evolves' with the discussion. This is also a way for core members to ensure that the process of integrating new features and major code changes is more "democratic" (WFc1, WFc2, SS2, WFc10). WFc1 says this transparency is vital to keeping their project "functional" (WFc2).

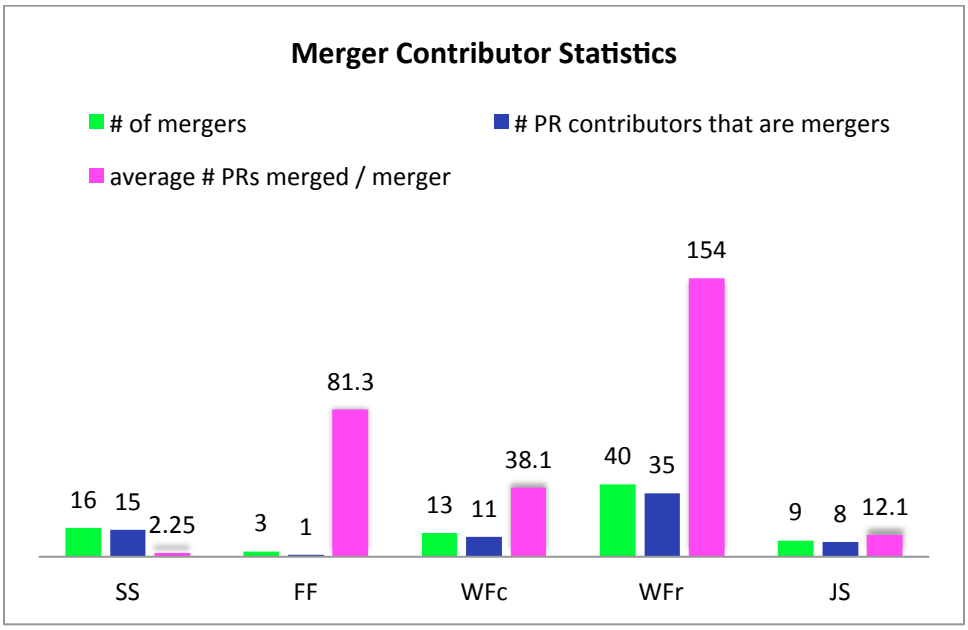


Fig. 2. Number of PR mergers; number of mergers that are also submitting PRs; average number of PRs pre merger

#### *6.2.4. Institutionalized Practice: Distribution of Merge Responsibility*

We find differences among communities with low or high merge rates in terms of merging behavior. Those among the communities with low merge rates we study are at either end of the extreme: they are either merging mostly their own pull requests or exclusively those of others. Those in communities with high merge rates are consistently divided; they merge some small percent of their own pull requests, as well as those made by other members who also have merge privileges and outside contributors who submit PRs.

*Low Merge Communities*

Responsibility for merging in SS is distributed among members who largely merge their own contributions. FF's community of mergers is dominated by two members responsible for merging other members' PRs but do not make PRs of their own; one member has made one PR which he merged (See Table 6.2). That is, FF does take contributions from outside contributors, but unlike all other communities, those with access are not submitting their PRs for community review. We are not aware of any other process for community review in place for those with access on FF. We were unable to interview this community and thus have no insights as to why they deviate from the other communities who employ PR for community review.

*High Merge Communities*

We find a more distributed model in communities with high merge rates, where a good number of mergers focus primarily on PRs that are not their own. Only maintainers and top mergers merge their own PRs, and even then, they do not merge all of them.

On WFc, we see that those who merge fewer pull requests are more likely to have their own pull requests merged by others. For instance, WFc4 has merged 11 PRs, 6 of which were other people's PRs, and 5 of which were their own. WFc4 only merges 1 in 3 of their own PRs. WFr is even more stringent (or experienced) it would seem; many of those who are merging have made a number of PRs, none of which were merged by them.

Like WFc and WFr, JS mergers do not merge all of their own PRs, but it is notable that the maintainer of the project has submitted 34 pull requests, only 4 of which he himself merged. JS has clearly embraced this model of community review.

## 7. Discussion

We found first that, counter to intuition, the size of a project has no impact on merge rates for the communities we studied. Second, the projects in which more pull requests are merged have more sustained members over time. Third, we find that communities with higher merge rates among contributors and higher contributor retention tend to have more distributed (non-centralized) practices for reviewing and processing pull requests. Specifically, in these communities (WFc, WFr, JS) members with collaborator access still process their work through PRs and encourage discussion around these PRs, which are usually not merged by the owner, but rather by some other member. This is in contrast to the two low merge rate communities; where members with the ability to merge code do not submit PRs or generally merge their own PRs.

We show that four of the five cases (WFc, WFr, JS, and to some extent SS) that we examine embody DL practices in various ways and to different degrees through a systematic operationalization of Gronn's three practices of distributed leadership. We also present clear contrasts among the five OSS projects on GitHub. We used DL as a lens through which to consider how communities may be more or less distributed in their practice.

The one remaining project, FF, does not present a **visible** distributed leadership model, and this is reinforced through the project's polite declinations to discuss their inner workings with researchers. The four remaining projects that do embody DL provide two contrasting models of leadership. A pattern of "distributed coordination" is exemplified by SS (Table 5.1). We see these differences most clearly through an examination of their practices around PRs, finding that they operate using GitHub for coordination work. WFc, WFr and JS, in contrast, use PRs more for collaboration.

| | SS | FF | WFc | WFr | JS |
|---|---|---|---|---|---|
| **Community Classification** | **Distributed Coordination** | **High Centralization** | **Distributed Collaboration** | | |

Table 5.1. Community Classification

Coordination on SS is central because informants from this project described their motivation for moving to GitHub as an attempt to reduce coordination costs to essentially "zero"; which is in fact what they claim to have accomplished (SS2). SS systematically takes advantage of the built-in transparency of the PR model to make the work of others visible but they do not adopt any apparent collaborative decision making model or

consistently engage in consensus-building debate around specific PRs. Content analysis of comments on SS might provide more insight into their practice.

The WFc, WFr and JS projects, in contrast, use PRs for consensus building activities, dissemination of project knowledge, and generation of insight. Responsibility for committing code is detached in most cases from the member who submits a pull request, even if that member technically has permissions to commit the code on their own. From a DL perspective, our informants describe this as a more "democratic" process. In the world of virtual OSS organizations, we find that the five GitHub cases we examine are clearly and richly differentiated from each other. Advancing DL as a mechanism for making sense of distributed OSS's requires operationalization of the factors described in our findings so that others might repeat and extend the findings we present.

| | SS | FF | WFc | WFr | JS |
|---|---|---|---|---|---|
| **% Contributors Merged** | Low Merge Community | | High Merge Community | | |
| **Repeat Contributors** | Low | | High | | |
| **Access Distribution** | High | Low | High | | |
| **PR Review** | Yes | No | Yes | | |
| **Distribution of responsibility for merging other peoples code** | Low | Low | High | | |

Table 5.2. Five Indicators Relevant to Distributed Leadership in OSS

We identified 5 factors that our data suggest are relevant in judging distributed leadership. These are listed in Table 5.2 as: Outside Contributors Merge Rate, Repeat Contributors, Access Distribution, PR Review and Distribution of responsibility for merging the PRs of others. We have no cutoff point, but one may be developed with a larger and more representative data set. While use of PRs by merging members with access to commit directly to the main branch as a form of "democratic" leadership is clearly important, we do not know how relevant this is as a metric of DL. A more extensive study is needed to conduct systematic evaluation of the emerging patterns.

Distributed Collaboration is the most similar to DL, though Distributed Coordination is, at the very least, an insular form of Distributed Collaboration – in the case of SS translates to a model that does not readily take in new members. It is not clear whether this pattern may hamper future research, but based on discussions with SS, they do not predict that their project will require much heavy maintenance in years to come. Part of this has to do with leadership vision, and, at some fundamental level, one's view of OSS. In contrast to distributed coordination, distributed collaboration is more consistent with notions of open source collaboration as a democratic endeavor, one in which the growth and maintenance of member contributors is critical to the success of a project.

## 8. Conclusions

Prior research on OSS development focuses on patterns of organization that are centralized and predictable across projects within a socio-technical context. The large-scale introduction of distributed software configuration management platforms like

GitHub, are hypothesized to foster more distributed forms of organization. Git logically suggests a greater distribution of patterns of leadership. Through the lens of DL, our study presents a number of specific organization types that suggest how such types could be rapidly identified, and which patterns of DL could potentially lead to greater engagement from newcomers. It is this newcomer engagement that we think is especially unique to GitHub, and worthy of continued study.

Future studies will extend to a large sample of projects on GitHub. We expect to measure the correlation between merge rates, retention and distribution of merge practices – for instance, what is the ratio of PRs made to those merged by the owner? In a more distributed community we expect to see mergers creating many more PRs than the number of PRs they merge by themselves. That is, in more distributed collaborations, other people should be responsible for merging PRs, regardless of whether they have access to commit (or "merge") capabilities. In these scenarios it makes sense that contributors with access are more poised to invite newcomers. It also makes sense that contributors who have early success stay on, and in some cases progress to becoming contributors with permission to commit.

Without comparison to projects hosted on other platforms, we cannot say that the features and interface of GitHub afford more distributed collaboration, but our data and analysis suggest that they may. Future comparisons will be valuable in determining the extent to which GitHub supports more distributed models. Our prior studies suggest this to be the case [1], and the findings presented here go further in explaining the practices and indicators of DL emerging on GitHub, and available through its API for designers, engineers and researchers.

#### *8.1.1. Limitations*

Our selection of only those projects that are currently popular on GitHub (based on the number of forks) represents a sampling bias. At the same time, we imagine that if these projects are visibly popular and thus successful (by definition) then other communities will be influenced by them. For instance, GitHub developers make very public their decision not to use git-flow. Other communities we studied are distinguished for their decision to only acknowledge PRs that are first published as issues, or for retroactively assigning a PR to a contributor. This is an exciting and evolving space that will be defined by trendsetters.

## 9. References


1. Author and Author. Pull Requests & Participation in Github. ((under review)).
2. Author. Group Informatics: A Methodological Approach and Ontology for Understanding Socio-Technical Groups. *JASIS&T Accepted*, (2012).
3. Crowston, K. A structurational perspective on leadership in Free/Libre Open Source Software teams. 2005.
4. Dabbish, L., Stuart, C., Tsay, J., and Herbsleb, J. Social coding in GitHub: transparency and collaboration in an open software repository. *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, ACM (2012), 1277–1286.

5. Ducheneaut, N. Socialization in an Open Source Software Community: A Socio-Technical Analysis. *Comput. Supported Coop. Work 14*, 4 (2005), 323–368.
6. Fielding, R.T. Shared leadership in the Apache project. *Commun. ACM 42*, 4 (1999), 42–43.
7. German, D. and Mockus, A. Automating the measurement of open source projects. *In Proceedings of the 3rd Workshop on Open Source Software Engineering*, (2003), 63–67.
8. Ghosh, R. and Prakash, V. The Orbiten Free Software Survey. *First Monday 5*, 7 (2000).
9. Gronn, P. Distributed leadership as a unit of analysis. *The Leadership Quarterly 13*, 4 (2002), 423–451.
10. Kim, A.J. *Community Building on the Web : Secret Strategies for Successful Online Communities*. Peachpit Press, 2000.
11. Krogh, G. von, Spaeth, S., and Lakhani, K.R. *Community, Joining, and Specialization in Open Source Software Innovation: A Case Study*. 2003.
12. Luther, K., Fiesler, C., and Bruckman, A. Redistributing leadership in online creative collaboration. *Proceedings of the 2013 conference on Computer supported cooperative work*, ACM (2013), 1007–1022.
13. Maass, W. Inside an Open Source Software Community: Empirical Analysis on Individual and Group Level. *University of St.Gallen*, (2004), 64–67.
14. Maass, W. Inside an Open Source Software Community: Empirical Analysis on Individual and Group Level. *University of St.Gallen*, .
15. Marlow, J., Dabbish, L., and Herbsleb, J. Impression formation in online peer production: activity traces and personal profiles in github. *Proceedings of the 2013 conference on Computer supported cooperative work*, ACM (2013), 117–128.
16. Mockus, A., Fielding, R.T., and Herbsleb, J.D. Two case studies of open source software development: Apache and Mozilla. *ACM Trans. Softw. Eng. Methodol. 11*, 3 (2002), 309–346.
17. Moon, J.Y. and Sproull, L. Essence of distributed work. 2000. http://www.firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/viewArticle/801/710.
18. Sanheim, R. Three Million Users. *The Github Blog*, 2013.
19. Scacchi, W. Understanding the requirements for developing open source software systems. *Software, IEE Proceedings - 149*, 1 (2002), 24 –39.
20. Scacchi, W. Free/Open Source Software Development Practices in the Computer Game Community. *IEEE Software 21*, (2004), 59–67.
21. Scacchi, W. Free/open source software development: recent research results and emerging opportunities. *The 6th Joint Meeting on European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering: companion papers*, ACM (2007), 459–468.
22. Thorpe, R., Gold, J., and Lawler, J. Locating Distributed Leadership. *International Journal of Management Reviews 13*, 3 (2011), 239–250.
23. Thorpe, R., Gold, J., and Lawler, J. Locating Distributed Leadership. *International Journal of Management Reviews 13*, 3 (2011), 239–250.
24. Zhang, W. and Storck, J. Peripheral Members in Online Communities. (2001), 7.

25. Github Has Surpassed Sourceforge and Google Code in Popularity. *ReadWrite*. http://readwrite.com/2011/06/02/github-has-passed-sourceforge.

**Appendix**

| SS | # Merges of PRs | # Merges of OTHER'S PRs (not theirs) | #PRs Made & Merged (merged by them) |
|---|---|---|---|
| Maintainer (SS1*) | 12 | 1 | 11 (11) |
| 1 | 4 | 0 | 5 (4) |
| 2 | 4 | 0 | 4 (4) |
| 3 | 3 | 3 | 0 (0) |
| 4 | 2 | 0 | 2 (2) |
| 5 | 1 | 0 | 1 (1) |
| 6 | 1 | 0 | 1 (1) |
| 7 | 1 | 0 | 1 (1) |
| 8 | 1 | 0 | 1 (1) |
| 9 | 1 | 0 | 1 (1) |
| 10 | 1 | 1 | 0 (0) |
| 11 | 1 | 0 | 1 (1) |
| 12 | 1 | 1 | 0 (0) |
| 13 | 1 | 0 | 1 (1) |
| 14 | 1 | 0 | 1 (1) |
| 15 | 1 | 0 | 1 (1) |

Table 6.1. Merge Behaviors for SS

| FF | # Merges of PRs | # Merges of OTHER'S PRs (not theirs) | #PRs Made & Merged (merged by them) |
|---|---|---|---|
| Maintainer | 153 | 153 | 0 |
| Maintainer | 90 | 90 | 0 |
| 1 | 1 | 0 | 1 (1) |

Table 6.2. Merge Behaviors for FF

| WFc | # Merges of PRs | # Merges of OTHER'S PRs (not theirs) | #PRs Made & Merged (merged by them) |
|---|---|---|---|
| Maintainer | 272 | 264 | 9 (8) |
| 1 | 124 | 124 | 1 (0) |
| 2 | 47 | 41 | 8 (6) |
| 3 | 24 | 24 | 0 (0) |
| 4 | 11 | 6 | 15 (5) |
| 5 | 4 | 2 | 3 (2) |
| 6 | 3 | 3 | 0 (0) |
| 7 | 3 | 2 | 1 (1) |
| 8 | 3 | 0 | 10 (3) |
| 9 | 1 | 1 | 17 (0) |
| 10 | 1 | 0 | 5 (1) |
| 11 | 1 | 0 | 3 (1) |
| 12 | 1 | 0 | 4 (1) |

Table 6.3. Merge Behaviors for WFc

| WFr | # Merges of PRs | # Merges of OTHER'S PRs (not theirs) | #PRs Made & Merged (merged by them) |
|---|---|---|---|
| 1 | 1024 | 1022 | 4 (2) |
| 2 | 410 | 410 | 0 (0) |
| 3 | 394 | 393 | 1 (1) |

| | | | |
|---|---|---|---|
| 4 | 386 | 386 | 89 (0) |
| 5 | 168 | 168 | 1 (0) |
| 6 | 152 | 151 | 89 (1) |
| 7 | 143 | 143 | 2 (0) |
| 8 | 134 | 134 | 45 (0) |
| 9 | 105 | 105 | 0 (0) |
| 10 | 88 | 88 | 20 (0) |
| 11 | 22 | 22 | 0 (0) |
| 12 | 18 | 18 | 0 (0) |
| 13 | 11 | 10 | 1 (1) |
| 14 | 8 | 8 | 3 (0) |
| 15 | 6 | 6 | 12 (0) |
| 16 | 3 | 0 | 31 (3) |
| 17 | 3 | 3 | 0 (0) |
| 18 | 2 | 0 | 4 (2) |
| 19 | 2 | 0 | 3 (2) |
| n = 21 | 1 per person | -- | -- |

Table 6.4. Merge Behaviors for WFr

| **JS** | **# Merges of PRs** | **# Merges of OTHER'S PRs (not theirs)** | **#PRs Made & Merged (merged by them)** |
|---|---|---|---|
| Maintainer (JS2*) | 47 | 43 | 34 (4) |
| 1 | 40 | 40 | 0 (0) |
| 2 | 11 | 9 | 3 (2) |
| 3 | 4 | 4 | 34 (0) |
| 4 | 3 | 2 | 146 (1) |
| 5 | 1 | 0 | 1 (1) |
| 6 | 1 | 0 | 42 (1) |
| 7 | 1 | 0 | 4 (1) |
| 8 | 1 | 0 | 62 (1) |

Table 6.5. Merge Behaviors for JS