Proceedings of the 52nd Hawaii International Conference on System Sciences | 2019

# Coordination in OSS 2.0: ANT Approach

Sangseok You
HEC Paris
you@hec.fr

Kevin Crowston
Syracuse University
crowston@syr.edu

Jeffrey S. Saltz
Syracuse University
jsaltz@syr.edu

Yatish Hegde
Syracuse University
yhegde@syr.edu

## Abstract

*Open source software projects are increasingly driven by a combination of independent and professional developers, the former volunteers and the later hired by a company to contribute to the project to support commercial product development. This mix of developers has been referred to as OSS 2.0. However, we do not fully understand the multi-layered coordination spanning individuals, teams, and organizations. Using Actor-Network Theory (ANT), we describe how coordination and power dynamics unfold among developers and how different tools and artifacts both display activities and mediate coordination efforts. Internal communication within an organization was reported to cause broken links in the community, duplication of work, and political tensions. ANT shows how tools and code can exercise agency and alter a software development process as an equivalently active actor of the scene. We discuss the theoretical and practical implications of the changing nature of open source software development.*

## 1. Introduction

Open source software (OSS) development[1] has traditionally been viewed by researchers as a collaborative outcome of individuals working independently. In this view, individual developers, dispersed around the world, work together toward a software product as a common goal. OSS development is interesting because it can be successful despite the absence of a formal structure and leadership in collaboration with dispersed individuals. The distributed nature of OSS development poses interesting questions and challenges, such as discontinuities in work, low coherence in work settings, and difficulties in developing shared mental models among participants [1]. Research in this tradition has shown, for example, that altruistic motivations are a major driving force that attracts individual developers and keeps them engaged throughout the project lifecycle [2, 3, 4].

More recently, there has been growing recognition that OSS development has been transformed by the involvement of actors other than individuals. Specifically, researchers are recognizing the important role of company teams that support a community and benefit from the collective work outcomes [5]. Especially in large OSS communities (e.g., Apache Spark or Hadoop), many individual developers are deployed to the community by their employer to represent and advance the company's interests. Fitzgerald referred to such transformation as OSS 2.0, with an emphasis on the impacts of company involvement [5].

We are interested in how the involvement of companies has changed the process and structure of collaboration in these communities. Despite recognition of these changes, research has not completely addressed the nature of the emergent processes, which may not be explained by our traditional view on OSS development as individual participation. In this paper, we focus on coordination in OSS development that embraces not only individual developers but also company teams. We define coordination as managing dependencies [6]. The addition of companies and company-sponsored teams engenders new levels of dependencies beyond the individual level, implying the need for coordination across individuals, teams, and organizations.

The addition of company teams also alters the roles of technological artifacts in OSS development. OSS development, as a largely distributed sociotechnical system, relies heavily on technological artifacts, such as communication tools, source code control systems, and the code itself [7, 8]. Although how distributed teams handle dependencies is well studied in the previous work, research still lacks evidence on coordination in OSS 2.0 [9, 1]. Moreover, by adding the corporate players to the scene, coordination in the recent OSS

[1] This movement is sometimes referred to as free/libre open source software to acknowledge the distinctive motives of the free software community.

URI: https://hdl.handle.net/10125/59538
ISBN: 978-0-9981331-2-6

development warrants an investigation with a new approach. Therefore, we pose two research questions for this study as follow:

1. *RQ1: How do individual developers, company teams, and companies coordinate in OSS 2.0?*

2. *RQ2: Given these different levels of interaction, how do technology artifacts support the coordination among the developers?*

To answer the research questions, we employ Actor-Network Theory (ANT), which addresses the interplay between the social and the technical in a sociotechnical system [10]. ANT is suitable to identify the emergence of new actors and the actor-networks across different levels of interactions. Besides, ANT treats objects equally as an actor [11, 10], thus providing helpful analytic perspectives from which to investigate the roles of technological artifacts.

By looking through ANT, we examined how dependencies were managed among participants at different levels and what roles technological artifacts played in OSS 2.0. We report findings from 20 semi-structured interviews with participants in several OSS communities. We discovered several kinds of actors in the OSS communities. Among those, company teams generated boundaries within themselves which engendered gaps in coordination across the community as a whole. Also, we found that artifacts like code were centerpieces of coordination by exercising agency over human actors. Therefore, a significant contribution of this paper is the taxonomy of several actors and their influence network for coordination across different levels mediated by artifacts.

## 2. Background

### 2.1. Coordination in OSS

OSS development teams have been studied by many scholars (e.g., see [12, 13] for reviews). They have been viewed as a unique context to study coordination in distributed work in particular [14]. OSS projects are characterized by its "open source" license, which allows inspection, modification, and redistribution of the source code of the software free of charge [1]. The unique "open" nature of OSS engenders numerous challenges in coordination among developers. For instance, OSS is often developed by distributed teams, whose team members are located around the world and meet infrequently, relying on communication tools such as videoconferencing, chats, and source code control systems [13, 14].

Crowston and colleagues [1] aggregated the collective effort in the research and built a theory to explain the coordination mechanism in such contexts. For instance, in their coordination theory, they highlighted that distributed developers manage dependencies through sociotechnical affordances of various artifacts. For example, documents, including source code and bug reports, can provide visibility and accountability of the current state of the work and inform developers of what can be done next, which is referred to as translucency [1].

Early OSS developments were often driven by voluntary individuals working independently [15, 14]. Thus, coordination in OSS has been understood mainly as how distributed individuals manage dependencies [1]. For example, Howison and Crowston [16] proposed superposition as a core mechanism of OSS projects, by which individual developers contribute small chunks of code (i.e., atomic commit) to the existing codebase that build on existing functionality.

However, recent trends in OSS projects demonstrate that company teams are playing much bigger roles in the development than before [9, 17]. Indeed, companies are shown to reduce development cost by *open-sourcing* the effort and become more innovative by engaging with OSS communities [3]. Germonprez et al. [18] found that software designers from companies can benefit from the engagement with OSS communities for more productive and creative design for their products. Also, Maenpaa et al. [19] viewed the OSS communities as a hybrid system that embed companies and governments as a part of a large software ecosystem.

Despite the increasing involvement of company teams in OSS projects, research still lacks empirical evidence on how the different kinds of actors in the OSS projects manage dependencies to turn the communal effort into a software product [20]. Part of the reason is the complexity in the collaboration among the different actors and stakeholders in OSS communities [20]. The addition of companies in OSS would change coordination because they often have different motivations and work styles not aligned with a given OSS community [3]. However, understanding the collaboration network in OSS projects are often incomplete or not include important actors [21, 16]. Thus, in this paper, the aim is at better understanding OSS projects by identifying important actors in the scene and mapping them in a complex coordination network.

### 2.2. Actor-Network Theory Approach

We employed an actor-network approach to understand coordination in OSS projects. Originated from science technology studies (STS), actor-network theory has been a useful lens to look through the relationships and dynamism in sociotechnical systems by illustrating their interests, desire, and agency (e.g., [22]). The ANT approach enabled sociotechnical scholars to discover new actors and their behaviors [11]. Another characteristic of the ANT approach is that the approach ascribes agency to objects and treats them as an equal actor in a given actor-network [10]. This allowed scholars to view technological artifacts as non-human actors and to focus on the technological affordances that enact agency [11].

We believe that the ANT approach is not only suitable but also useful for studying OSS projects for several reasons. First, given the increasing presence of company teams in OSS projects, the ANT approach allows us to identify them as a new actor and highlight their behaviors and agency. Second, although previous studies emphasized the role of technology artifacts, such as source code control systems and communication tools, in open collaboration, they were viewed only as tools that convey social cues [7] rather than as actors in their own right, which is the perspective of ANT. Lastly, there is dearth of studies that employed the ANT approach to explain coordination in OSS projects. One exception is Ducheneaut [22], which illustrated socialization processes of new participants in an OSS community.

## 3. Method

### 3.1. Data Collection

We conducted semi-structured interviews to obtain an in-depth understanding of coordination in OSS communities. Our research team recruited a total of 20 developers from several OSS projects across communities, including some large communities like Apache, Mozilla, and Linux. Because of the exploratory and inductive nature of the study, our goal was to develop a diverse sample of interviewees that spanned many contexts, rather than a representative sample of some population.

The recruitment of the subjects was two-fold. In the initial stage of the study, we employed a snowball approach by getting referrals from relevant conferences (e.g., ApacheCon). Four subjects were recruited at this stage. These initial points of contact were used to develop and pilot test the interview protocols and to identify major actors in OSS communities (e.g., contributors, committers, and project management committees) and tools (e.g., GitHub, JIRA, and emails).

In the main part of the study, we posted recruitment solicitations to developer mailing lists of projects with varying degrees of community size, activity levels, and history. Sixteen subjects were recruited through this method. The interviews were conducted based on the protocol developed from the initial interviews, which included questions regarding subjects' professional background, general descriptions on OSS projects they contributed, tools used for individual and teamwork, and how they managed dependencies within and outside their team. Each of these interviews lasted about an hour either in person or online. Interviews were audio-recorded and then transcribed.

Our data are diverse regarding subjects' tenure in OSS communities, employment type, and projects. Details of the subjects are shown in Table 1. Our subjects were involved in the following OSS communities: Apache projects (Jena, Stanbol, Clarezza, Fedora Commons, Islandora, Lucene,Spark, Hadoop, HBase, ZooKeeper, BookKeeper, Hive, Derby), iPython Notebook Extention, Jupyter Environment Kernels, Pypandoc, Pandas, Riot.js, Drupal, TensorFlow, Samvera, Open ONI, OpenCV, Meme generator, VMware vSphere Integrated Containers, and jQuery Plugins. To provide anonymity, we do not show subjects' specific affiliations and in quotations, we use pseudonyms in place of the names of individuals and companies.

### 3.2. Data Analysis

The transcripts were coded based on a grounded theory approach to identify themes emerging across the subjects [23]. The initial coding was an iterative process as we tried to identify themes related to actors, their interactions and networks, and patterns of them. Through this process, we discovered more than 140 themes. We then used the open coding to obtain 18 high-level categories and generate running hypotheses. Our research team discussed consistently throughout the data analysis. The analysis continued until we believed that the data was theoretically saturated.

## 4. Findings

In this section, we first describe several kinds of actors, which include human actors and non-human actors. We then illustrate the actor-networks, as emergent relationships of the actors, to unpack the coordination mechanism in the OSS communities.

Table 1. List of Study Subjects

| Subject | OSS Experience | Full Time |
|---|---|---|
| P1 | 15 years | Yes |
| P2 | 3 years | Yes |
| P3 | 4 years | Yes |
| P4 | 12 years | Yes |
| P5 | 3 years | Yes |
| P6 | 6 years | Yes |
| P7 | 5 years | Yes |
| P8 | 7 years | Yes |
| P9 | 12 years | Yes |
| P10 | 1 year | Yes |
| P11 | 6 years | No |
| P12 | 7 years | Yes |
| P13 | 2 years | Yes |
| P14 | 8 years | Yes |
| P15 | 5 years | Yes |
| P16 | 5 years | Yes |
| P17 | 1 year | No |
| P18 | 19 years | Yes |
| P19 | 3 years | No |
| P20 | 2 months | No |

## 4.1. Human Actors

In ANT, an actor, derived from a semiotic definition of *actant*, is defined as an entity that acts or grants other's activity [10]. In that an actor can be literally anything if it is the source of an action [10], actors can include both human and non-human entities. For human actors, we identified that there are three categories in multiple levels: individuals, company teams, and organizations (i.e., companies). The three categories are in a hierarchy, in which an individual actor may be affiliated with a company team in an organization. The reason why we highlight these actors distinctively is that actors in each level are found to manifest unique influences on the project based on their interests, motivations, and resources at different levels.

**4.1.1. Contributors and Committers** Following the definition by Apache Software Foundation (ASF[2]), a contributor[3] is "anyone who wants to contribute (code, documentation, tests, ideas, anything!) to any project." All subjects were a contributor at least in one OSS project. In our data, code and feature ideas were more prominent than other types of contribution. Most participants in an OSS project begin as contributor. Our subjects identified themselves as contributors based on the amount of effort put forth and the size of the changes in code accepted and merged to the system. P20 had a clear role perception: *"I think my role was to code for the project. I am the contributor, I would say. Whatever ideas they give, I have taken inputs and I made sure it's gonna be user friendly to them. I would say I'm a contributor."* P9 also identified himself as a contributor in one project by *"help[ing] with actual code, identify[ing] bugs, and help[ing] to fix those bugs."*

On the other hand, committers are individuals who have permissions to make changes to a shared source code of an OSS project[4]. Their rights regarding the source code involved both writing code and accepting changes from contributors submitted through code repositories (i.e., making commits). In our data, there were eight subjects who were committers in at least one OSS project: P1, P2, P4, P5, P7, P9, P12, and P14. The committers in our data had more substantial involvement and exercised more power in their projects than contributors did,, by fulfilling several responsibilities: selecting issues to solve, reviewing changes submitted by contributors, accepting and rejecting the changes, writing code by themselves, and communicating with other developers (P2, P4, and P5). As P9 described, committers handle *"virtually every one of the pull requests that goes into the project, I'm either committing myself because I wrote it and somebody else is reviewing it, or vice versa"*.

These roles were not exclusive in a person, and those who work for multiple projects may have different roles by project. For instance, P18 has been working in OSS projects for 19 years, and his role involved being on the project management committee (PMC) for an Apache project while still contributing code to another OSS project as an individual contributor. The multitude of involvement in projects is common across our data and is more prominent among people who have longer tenure in the OSS communities (e.g., P1, P3, P4, P9, P11, P14, and P18). P3 mentioned having membership in multiple projects: *"The [Project A] is the one I'm heavily involved in and the [Project B] is the one that I'm a small time contributor for."* P11 was also involved in several projects and explained different levels of ownership and involvement: *"There have been some projects, which I have worked in small teams, but here, most of them are my own projects."*

**4.1.2. Company Teams** In this study, a company team refers to a group of people in a company who work together on the same module of a software product.

[2]https://www.apache.org/foundation

[3]Contributors are also referred to as developers in some cases (e.g., Apache Software Foundation), but we use the term *developers* as an inclusive term for individuals who are involved in OSS projects, regardless of the level of contribution and rights.

[4]https://en.wikipedia.org/ wiki/Committer

Our subjects had responsibilities for a component of a software product, which was closely related or benefited from the collective effort of development in an OSS community. Thus, not all members of a company team were involved in the project to the same degree of time and the amount of code contribution. A company team in this study should be conceptually distinguished from a network of people collaborating on the same module of an OSS project with a perceived team membership but without a common affiliation. Among twenty subjects, sixteen indicated that they were paid to work on an OSS component of a product of their organization (see Table 1). The other four subjects were not affiliated with a for-profit organization and contributed to OSS communities voluntarily.

Team size varies from small with 3-4 people (e.g., P2, P10) to large with 7-8 people (e.g., P8, P13, P16). Our subjects were a point of contact between the companies and the projects. They were more active in the OSS community than other team members and worked on the front line for their team in the community. For instance, P10 mentioned different engagement with OSS communities by team members: *"It's normally just two, or three. ... There will be one actually from the team working closely with me and then some of the other guys, it's just whenever we have some questions we just randomly discuss and occasionally seeking for their help."*

Company teams had internal meetings to determine what to do for their product. In some cases, company teams employed the Agile methodology for their product development by having multiple scrum teams whose members worked closely in OSS communities (P1, P9, P13, and P10). The internal meetings included discussions on what to do, scheduling, and reviewing code before submitting to the OSS codebase. Thus, contributions by the company teams were aligned with the company's technical interests and resources such as time and manpower (P1, P10, P18). P10 explained the internal communication with team members outside the OSS community: *"Our main goal is not to make contributions, but we make contributions during our process or progress. We definitely have some discussions with team members because you need to let them know we need to purchase some time on contributing and what we will contribute. So basically the discussions happen all the time."*

Companies are relatively less visible, but still appeared to play big roles behind the scene. Companies have been viewed as sponsors of OSS projects, but in our study [9], they seemed to exercise more agency than sponsors in the background. So, we treated them as an entity that possess their own collective characteristics and intentions. For instance, teams and companies perceive ownership of certain module of an OSS project in case the module is critical to their product. An experienced contributor/committer P1 provided an illustrative example: *"Often times different subsystems are owned by different parts of whoever is the ... like in the case of [Product A], that's owned by [Company A], they have employees who work on it full time and they have their own division of work."*

## 4.2. Non-Human Actors

Code and tools were identified as significant non-human actors in OSS communities. They emerged as integral parts in understanding coordination in OSS communities. Code itself was found to be a conduit that delivered signals on what needed to be done and illuminated the power dynamics among human actors. Tools and documents produced during the use of them became actors based on their technical affordances and functionalities, such that pull requests and notifications enabled highlighting code changes and discussions among developers.

**4.2.1. Code** Code itself was found to play significant roles in coordination in OSS communities. On the one hand, code was the main artifact as an outcome of the collective effort in an OSS community. On the other hand, we discovered that the code acted through its affordance of delivering meaningful signals for coordination.

First, code changes signified what needed to be done and implied the layers of contribution by the community members. By embedding bugs and errors, code was the immediate source for contributors to find an area to work on. We discovered that starting a contribution from reading the codebase was particularly helpful for low-hanging fruits like small bugs and typos identified directly from the code, to be corrected (P2, P3, P9). P1 identified himself as a *"code janitor"*, which *"hooked me into the project and got me going"*. Along with source code control systems, code was an indispensable actor by containing histories of the past work and awareness of what was being done.

Furthermore, the code filled a role in delivering cues on the work and stimulating actions from developers by the way it is written. This made the code a major actor in the OSS communities and the one of first places where developers went for coordination. For instance, P9 mentioned the self-sufficiency of the code and its usefulness for next actions: *"[T]he code itself is not well-documented anywhere except in the code. You do have to get in there and trace through the code to figure*

*out what's going on. If there's something you need to change, you have to trace through the code to figure out how to change it."*

Second, structures and norms to write code varied by OSS community, and the code enforced contributors to follow the community protocol (P4, P5, P11). P11 defined: *"Good code is well commented, it is modularized, so just by looking at the functions, function names, it's written beautifully in a way thats well indented, the variable names are nice, the use of underscores and capital letters, everything is good."*

Interestingly, the norm to write good code was acted differently between committers and contributors. This seemed related to the power that committers possessed over contributors, such as prioritizing, filtering, and selecting issues to solve and approve to merge. Although committers were expected to write better code for projects, contributors believed that committers were free from the good code requirements due to the bigger power. P4 said: *"As a committer I can write some shit, and commit it. Which is where I think a lot of the crap code comes from. Contributor code is a small set of the overall codebase, because committers write more code than contributors, but I bet it's the better code on average."*

Third and the most important, the malleable nature of the code enabled the collaboration among the developers through the loosely structured coordination. In OSS communities, code was a public, collaborative artifact, which was viewed as an organism that grew and evolved over time by embracing mistakes, errors, corrections, and practices from different people with unique interests and skills. P16 summarized the *"code malleability"*: *"All software has a number of bugs even the commercial ones. But one of the benefits of using OSS is that we can look at the source code to see the cause of a problem and be able to fix it. So, I think it is not about how complete the software is but the important thing is the accessibility to the source code."*

**4.2.2. Tools** Subjects reported on many tools that shaped OSS development. First, Jira and similar tools provide various collaborative development functionalities, including issue tracking, bug tracking, and project management features. Jira was the most popular tool observed in our data: larger and mature projects like Apache Spark and Hadoop relied more on Jira to manage issues and communicate among developers.

Issue tracker was the core functionality in Jira, where contributors suggested an issue (i.e., bug fixes and new features) to the community. P3 emphasized the importance of the issue tracker as a starting point of a contribution that provides awareness of other developers' activities: *"I think that's always the best place to start. It is like, look at what other people have found and see if there is anything that interests you that you think fits with your skill level or your level of comfortability with the software and go from there."* However, despite the heavy use of the issue tracker in Jira, subjects also expressed that the overwhelming volume of issues stacked up and were not effectively controlled by the limited number committers (e.g., P3, P4, P6, P8, P15), which resulted in uneven distribution of committer attention to the issues.

Next, GitHub was gaining presence in projects in our data. Smaller and younger projects were adopting GitHub for its advanced git (a source code version control system) management. GitHub also provide a communication medium, e.g., pull requests, through which a contributor submitted a code change for a review (P2, P7, P10, P11, P14, P17, and P19). Pull requests were found to be central in the use of GitHub. They provide a notification of changes. They are a major communication channel between committers and contributors and a place to discuss the submitted changes with other developers. GitHub allowed developers reviewing a pull request to comment in line, tag a specific person, and easily visualize differences in code. P6 described the pull requests for communication: *"They [committers] do comment on pull request and yeah, it's good way to like, really tell the submitter I don't like this code or correct this way and resubmit it or maybe this part of code you need to create a separate pull request. [T]hat kind of discussions happen on pull requests."*

## 4.3. Duality of Developer Identity

All but four of our subjects were members of a team affiliated with a company. Subjects from company teams were deployed to an OSS project by their employer, who was thus a stakeholder in the development community. The interviews show that the individual contributors had duality in identity, such that they participated in the community as an individual, while they represent the interests of their company team and the company. P8 described: *"I think you know you're part of the open source community, you wear that open-source hat but still you do represent for you at least are driven by your company interest somewhere."*

Thus, in determining what to do for the community, contributors were governed by two different sets of concerns: organizational decisions and individual interests. P3 reflected the time when he was working for a company: *"I was told what to do and it was until I left*

*and actually worked for the foundation, then I had the freedom that I have to figure out like, "Okay, like what next?"*. Decisions regarding certain code changes and feature implementations were made in internal meetings within the company team. An individual contributor was a point of contact for each team. They conveyed the team's and company's needs for a feature to the community and implemented them by working with other developers in the company and the community. P16, as a team leader, emphasized: *"I am a decision maker so basically I assign task to each team member based on his/her strengths. They will see their assigned tasks in issue tracker."*

On the other hand, the company-affiliated contributors retained agency to make small changes bypassing explicit communications within their team. Fixing bugs and typos are good examples of the small contributions. P1 said:*"That's not the right work but just looking for sort of looking for little minor, tiny little improvements because I like the project and I wanted to be engaged with it."* P6 explained the duality more specifically: *"If I am doing some fun work for me, right. So that's a different story where I do some fun work. But mostly it's not that case, mostly it's driven by my current needs in the organization and that open source stream that had a gap in that."*

In contrast to contributors, it was observed that committers had more agency to choose what to do and thus made changes to the project based on their greater power. However, the greater agency of committers was not independent, but still influenced by their company as the contributors were. More specifically, while contributors' activities were dictated more by their employer's needs with some space for their personal interest, committers were able to exercise more autonomy with consideration of the expectation of their company. P1 summarized this well: *"[A]s I become more central to the [project], I'm a committer now and I'm on the project management committee, I feel like I am one of those much more central people. I tend to just ask myself what do I feel like working on? What is interesting to me and what would be helpful to my employer obviously as well. That's in my mind as well, which is one of the reasons they're happy to pay me to do this."*

As such, many individuals had multiple memberships in teams in and outside their employer and roles acquired based on the amount of code contribution and the tenure in the community. We discovered that the membership of company teams resulted in a duality when exercising their agency as an individual participant of a community. The agency was performed mainly by deciding what to do and how to do implement the decision. Such decisions were made through a developer's motivation to contribute to the codebase, but company teams behind the developers influenced the decisions by enforcing technical demands and needs for developing commercial products.

### 4.4. Actor-(Broken) Network

When connecting the actors of company teams into an actor-network, we discovered that the whole actor-network was divided into multiple clusters based on companies. However, the coordination among the clusters was reported to be not effective. Interviewees did not report on formal connections among the company teams in coordinating works in the community. Instead, coordination was done via loose ties between individual developers representating of their teams. Their interactions determined what should be done for a project (e.g., adding a feature or deciding a long-term direction of the project). Subjects in larger projects reported the disjoints between company teams. For instance, P13 mentioned *"It is more common to have internal meetings with people who are working for the same company. It gets harder for people outside the team to follow what is going on for the project and what they are trying to do. There is like a big void that you can't never see. No discussion, it's like something is done all of sudden like Ta-da."*

The broken network was reported to cause coordination problems. One problem was the duplication of effort in writing code for the same module by individuals from different company teams. The duplication of effort was largely viewed as a negative thing (i.e., *"waste of time and effort"* (P12)). These problems were in part because of 1) lack of explicit coordination on assigning a task among developers from company teams and 2) the tools failing to provide social awareness. P15 explained the disjoints in coordination between company teams via individual developers: *"Most people would prefer to work with the team or person that they can easily communicate like physically co-located or someone they know. This is one of the disadvantages of open source development, how can you know when a person is going to finish or will he ever finish it. So, working on the same module, its more likely to be a group of people who are physically working together."*

In addition, the broken ties may hamper the overall development process of a project. Company teams put forth their interests, tried to optimize their product, and prioritized the features that were wanted by their customers. Thus, rather than negotiating an end goal of the project with other parties, the company teams

often became possessive of their code and even prevent others' work from being used widely in a community. P2 explained: *"Usually people from those companies start coming and pressuring to get the work done. That's what usually breaks ties I would say, in my experience, on the bigger contentious things if they just don't choose to fork off and make their own component. They were essentially doing everything they could to prevent us from performing well on their clusters."*

## 5. Discussion

We described human and non-human actors and the actor-network. The human actors include individual developers, company teams, and companies, while tools and code itself were identified as important non-human actors. These actors failed to establish one whole actor-network, but rather had broken links and inefficiencies in coordinating works among them.

### 5.1. Actor-Network in OSS 2.0

The actors constituted a multi-layered actor-networks for coordination in OSS communities (see Figure 1). The actor-network consists of both the human and the non-human actors identified above and illustrates relationships among the human actors, which are mediated by the non-human actors. In addition, the actor-network identifies company teams as major players, with which individual contributors and committers are affiliated. Company teams themselves are the actors, but at the same time, hold its own assemblage embracing individual developers and team members who are outside an OSS community. In the actor-network, company teams established a boundary based on corporate affiliation (depicted with solid lines in the figure), whereas OSS communities were intended to be *open* with *"low barriers of entry"* (P4) from outside (the dashed line in the figure). Overall, the actor-networks in the OSS communities were found to be collections of heterogeneous actors who exchanged influences to attain mutual interests out of coordination.

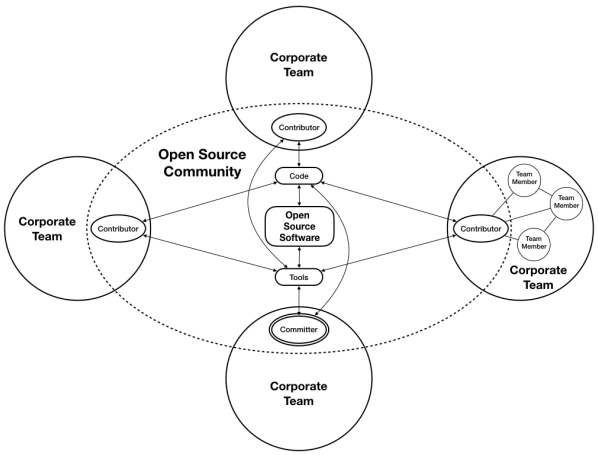


**Figure 1. Actor-Network in OSS 2.0**

### 5.2. Theoretical Implications

**5.2.1. Code with Agency** ANT allowed us to attribute more agency to non-human actors. In particular, code itself turns out to be the centerpiece of the coordination among the human actors in OSS development. The codebase was a useful conduit to deliver signals regarding what to do, what has been done, and who is responsible. These signals had an active role to change behaviors of individual developers, team meetings, companies, and eventually the direction of the project development throughout the community.

In addition, the codebase contained rich information about an OSS community. We found that communities have their own rules and norms in writing code and submitting changes. Writing good code and commit messages were implicitly defined within a community and enforced to individual contributors. Code also manifested the tension between company teams who prioritized their interests.

Tools like pull requests and issue trackers were supporting the agency of code by easing the effort of writing, reading, and making sense of the codebase. Technical affordance of the tools, including notifications and commenting in line, provided social awareness and contextual continuity to developers working in different time and locations.

Theoretically, we understood the role of non-human actors as a mediator. *Mediators* "transform, translate, distort, and modify the meaning or the elements they are supposed to carry" [10, p. 39]. Mediators should be distinguished from *intermediaries*, which simply tranfer meaning and information without altering the actor's behaviors, intentions, and expectations [10]. Therefore, by posing our non-human actors as mediators, we highlight its robust agency to influence the human actors and eventually the coordination in the OSS communities. Our study provides a new insight into the current literature on OSS work by highlighting the more significant role of the code. Previous works on OSS 1.0 (e.g., [7]) mainly focused on the technical affordance of collaboration tools and treated the code as an artifact or work outcome. However, we showed that

the codebase, as a malleable, collective organism, is at the core mechanism of coordination in OSS community.

Understanding how different actors react and behave related to code should be an interesting theoretical concern. Our data showed that committers and contributors had different attitudes and practice in writing and interpreting code. The former was reported to have more freedom to choose which part of the code to work on, whereas contributors were required to write better code that should also easily make sense (e.g., P4). Such different relationships of code with actors may raise tensions and illuminate several *matters of concerns* as Latour noted [10].

**5.2.2. Companies as New Actors and OSS 2.0** Companies contributed to the projects by deploying individual developers from a team to an OSS community, and they leverage the collaboration between their employees and the community. The individuals from company teams were working as a conduit to provide their resources to the community and getting a community of help for their product. However, our data implied that different interests of companies did not have a clear forum for a communal resolution and that there was a lack of translucency among company teams. These seemed to hinder individuals from working together.

By considering companies as actors, our understanding of other roles can change. For instance, given that committers possess more agency and power to determine what issues to solve and what changes to merge, companies should want their employees to become committers. Also, it may become harder for purely voluntary individuals to become a committer. Indeed, the enforcement of norms about code quality—potentially in conflict with the norm of keeping *"the barrier to entry on the open sourcing low"* and welcoming more contributors (P4)—could be viewed as a way to enhance company control.

Moreover, putting companies forth may alter how OSS development works in the future. Most OSS communities, especially in Apache Foundation, adopt *"the consensus-based, community-driven governance"* (a.k.a., The Apache Way). Our findings suggest that we may have to reconsider the agency of individual participants in OSS development, in conjunction with the agency of corporate stakeholders. This study not only echoes the notion of OSS 2.0 by Fitzgerald [5] but also puts more emphasis on the agency of companies in the OSS scene.

We describe the roles of companies, which were relatively unseen in the network [3, 20]. Putting the company teams and companies forth can provide new insight into understanding the recent trends in OSS communities. Previous literature on OSS projects primarily viewed that projects were driven by a collective effort of independent individuals [16]. However, especially in big projects, companies are at the core by providing resources, such as manpower and supporting conferences, and expanding the user base of the project.

Although the presence of companies in OSS scenes has been noted in the previous works, research identified companies as a type of patron or supporter of a community [9, 19]. However, we unfold the underlying relationships of company teams and concerns of individuals who are involved in and work with the company teams. Specifically, individuals from company teams still voluntarily exert effort, decide what to do, and negotiate between their interests and those of their company. We highlight such dual-facet of the individual developers in OSS 2.0.

ANT was useful to illuminate the other side of the coordination across individuals, company teams, and companies in OSS communities. Because ANT does not limit our observations to predefined groups and their roles (e.g., only contributors and committers) and allows us flexibility of describing emerging relationships [10]. Our findings were discovered largely through paying attention to relational aspects (e.g., how they work *together*, how code *changes* behaviors), rather than actions of each actor (e.g., how they *use* tools).

## 5.3. Practical Implications

We showed that code has agency and delivers various signals for coordination. Our practical implications, thus, emphasize the increasing role of code and its supporting tools. First, tools for writing and submitting code should provide more nuanced social awareness. We found that there were certain norms and rules to write good code to be reviewed by committers. The tools can embed a community-approved template including such information as the writer, the logic behind, and their affiliation.

Second, tools should support modularity in the codebase. Our findings suggest that companies seem to work on different part of code, but the boundaries among the parts were not clearly negotiated nor defined among corporate stakeholders. Tools can improve coordination among the company teams, for example, by showing particular lines that are affected by new commits and providing enhanced searchability. It can also have a dicussion forum for companies. In those ways, work can be shared without version conflicts and avoid duplication of effort by different parties.

### 5.4. Limitations

This study has several limitations. First, we were not able to examine the development and evolution of actor-networks. Such matters may require other types of data, such as longitudinal observation of online activities and code changes. Second, as we analyzed data, we realized that other parties, such as customers of a product based on an OSS project and users of an OSS project, may play significant roles as well. Future studies should involve these actors as well and complete the actor-network.

## 6. Conclusion

We looked at the coordination in OSS communities through the lens of ANT approach. We discovered several kinds of human and non-human actors. Companies were identified as a significant actor in the scene, which have been treated tangentially in OSS communities. We also highlighted the active roles of tools and code as mediators, through which social cues, norms, and information on the work were delivered. Our results inform the design of tools supporting coordination in OSS communities, and suggest several ways to improve transparency in collaboratively writing code.

## References


[1] K. Crowston, C. Østerlund, J. Howison, and F. Bolici, "Work features to support stigmergic coordination in distributed teams," in *Academy of Management Proceedings*, p. 14409, Academy of Management, 2017.

[2] K. J. Stewart and S. Gosain, "The impact of ideology on effectiveness in open source software development teams," *MIS Quarterly*, vol. 30, no. 2, pp. 291–314, 2006.

[3] M. Andersen-Gott, G. Ghinea, and B. Bygstad, "Why do commercial companies contribute to open source software?," *International Journal of Information Management*, vol. 32, no. 2, pp. 106–117, 2012.

[4] G. Von Krogh, S. Haefliger, S. Spaeth, and M. W. Wallin, "Carrots and rainbows: Motivation and social practice in open source software development," *MIS Quarterly*, pp. 649–676, 2012.

[5] B. Fitzgerald, "The transformation of open source software," *MIS Quarterly*, pp. 587–598, 2006.

[6] K. Crowston and J. Howison, "The social structure of free and open source software development," *First Monday*, vol. 10, no. 2, 2005.

[7] 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*, pp. 1277–1286, ACM, 2012.

[8] F. Bolici, J. Howison, and K. Crowston, "Stigmergic coordination in FLOSS development teams: Integrating explicit and implicit mechanisms," *Cognitive Systems Research*, vol. 38, pp. 14–22, 2016. Special Issue of Cognitive Systems Research – Human-Human Stigmergy.

[9] J. Lindman and I. Hammouda, "Support mechanisms provided by FLOSS foundations and other entities," *Journal of Internet Services and Applications*, vol. 9, no. 1, p. 8, 2018.

[10] B. Latour, *Reassembling the Social: An Introduction to Actor-Network-Theory*. Oxford University Press, 2005.

[11] N. Kumar and N. Rangaswamy, "The mobile media actor-network in urban India," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 1989–1998, ACM, 2013.

[12] A. Aksulu and M. R. Wade, "A comprehensive review and synthesis of open source research," *Journal of the Association for Information Systems*, vol. 11, no. 11, pp. 576–656, 2010.

[13] 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 Computing Surveys*, vol. 44, no. 2, pp. 7:1–7:35, 2012.

[14] J. D. Herbsleb and R. E. Grinter, "Splitting the organization and integrating the code: Conway's law revisited," in *International Conference on Software Engineering (ICSE)*, pp. 85–95, ACM, 1999.

[15] K. Crowston, Q. Li, K. Wei, U. Y. Eseryel, and J. Howison, "Self-organization of teams for free/libre open source software development," *Information and Software Technology*, vol. 49, no. 6, pp. 564–575, 2007.

[16] J. Howison and K. Crowston, "Collaboration through open superposition," *MIS Quarterly*, vol. 38, no. 1, pp. 29–50, 2014.

[17] A. Bonaccorsi, S. Giannangeli, and C. Rossi, "Entry strategies under competing standards: Hybrid business models in the open source software industry," *Management Science*, vol. 52, no. 7, pp. 1085–1098, 2006.

[18] M. Germonprez, J. E. Kendall, K. E. Kendall, L. Mathiassen, B. Young, and B. Warner, "A theory of responsive design: A field study of corporate engagement with open source communities," *Information Systems Research*, vol. 28, no. 1, pp. 64–83, 2016.

[19] H. Mäenpää, F. Fagerholm, M. Munezero, T. Kilamo, T. J. Mikkonen, *et al.*, "Entering an ecosystem: The hybrid OSS landscape from a developer perspective," in *CEUR Workshop Proceedings*, 2017.

[20] G. J. Link and D. Jeske, "Understanding organization and open source community relations through the attraction-selection-attrition model," in *Proceedings of the 13th International Symposium on Open Collaboration*, p. 17, ACM, 2017.

[21] E. von Hippel and G. von Krogh, "Open source software and the private-collective innovation model: Issues for organization science," *Organization Science*, vol. 14, no. 2, pp. 209–223, 2003.

[22] N. Ducheneaut, "Socialization in an open source software community: A socio-technical analysis," *Computer Supported Cooperative Work (CSCW)*, vol. 14, no. 4, pp. 323–368, 2005.

[23] K. Charmaz and L. L. Belgrave, "Grounded theory," *The Blackwell Encyclopedia of Sociology*, 2007.