AN ACTOR-NETWORK APPROACH TO COORDINATION IN OPEN SOURCE SOFTWARE 2.0

Sangseok You*
Department of Information Systems and Operations Management
HEC Paris
you@hec.fr

Kevin Crowston
School of Information Studies
Syracuse University
crowston@syr.edu

Jeffrey Saltz
School of Information Studies
Syracuse University
jsaltz@syr.edu

Yatish Hegde
School of Information Studies
Syracuse University
yhegde@syr.edu

* Corresponding Author

AN ACTOR-NETWORK APPROACH TO COORDINATION IN OPEN SOURCE SOFTWARE 2.0

ABSTRACT

Open source software is increasingly driven by a combination of independent and professional developers, the former volunteers and the later hired by a software 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 coordination spanning individuals, teams, and organizations in OSS 2.0. Using Actor-Network Theory (ANT), we describe how coordination and power dynamics unfold and how technological artifacts both display actions and mediate coordination efforts. Internal coordination within an organization was reported to create competing networks against the network for the whole OSS community by breaking the alignments of interests. ANT shows how software development tools and code, as active actors, exercise agency in attracting developers to work on problems and informing the layers of collaboration. We discuss the theoretical and practical implications of the changing nature of OSS.

Keywords: OSS 2.0, open source software, actor-network theory, ANT, and coordination

INTRODUCTION

Open source software (OSS) development[i] was originally viewed by researchers as a collaborative outcome of individuals working independently [5]. 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 is an example of collaboration with dispersed individuals that is successful despite the absence of a formal structure and leadership. The distributed nature of OSS development poses interesting questions about how it addresses challenges such as discontinuities in work, low coherence in work settings, and difficulties in developing shared mental models among participants [40,43]. Research in this tradition showed, for example, that altruistic motivations are a major driving force that attracts individual developers and keeps them engaged throughout the project lifecycle [2,43,45].

More recently though, there has been growing recognition that OSS development has been transformed by the involvement of participants other than individual developers. Several companies, including both software firms such as Microsoft and large technology-driven commercial enterprises such as Walmart, use open source software for their business and make their software open source for anyone to benefit from it [22]. In 2018, IBM announced that they would acquire GitHub, one of the most popular source code control and collaboration platforms for open source developers and companies [23].

Researchers also recognize the increasing role of companies that support a community and benefit from the collective work outcomes [5,11,24]. In large OSS communities (e.g., Apache Spark, Hadoop, and TensorFlow), many developers are deployed to the community by their employer to represent and advance the company's interests. Such transformation was referred to as OSS 2.0, with an emphasis on the impacts of company involvement [24].

We are interested in how the involvement of software companies has changed the coordination and the structure of collaboration in these communities. Despite the growing recognition of these changes, research has not completely addressed the nature of the emergent coordination network, which may not be explained by our traditional view on OSS development as a collaborative outcome of individuals' voluntary participation.

In this paper, we focus on coordination in OSS that embraces not only individual developers but also company teams. We define coordination broadly as managing dependencies [13,14]. Coordination thus encompasses how various entities in open source communities, such as individual developers, company teams, and their members, align their interests, desires, and actions among themselves during OSS development [13,40]. The addition of software companies and company-sponsored teams engenders new levels of dependencies beyond the individual level, implying the need for coordination across teams and organizations [5,24].

The addition of company teams also alters the roles of technological artifacts in OSS development [11,41]. 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 [4,44]. Although how distributed individuals handle dependencies is well studied in general, research still lacks evidence on coordination in the context of OSS 2.0 [14,35]. Moreover, by adding the corporate players to the scene, coordination through artifacts in the recent OSS development warrants an investigation with a new approach [44]. Therefore, we pose two research questions:

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

RQ2: How do technological artifacts influence coordination in OSS 2.0?

To answer the research questions, we conducted interviews with developers from a range of OSS projects. We analyzed this data through the lens of Actor-Network Theory (ANT), which addresses the interplay between the social and the technical in a sociotechnical system [33]. ANT is suitable to identify the emergence of actors and actor-networks across different levels of interactions. Besides, ANT treats objects equally as an actor [6,28,33], thus providing valuable analytic perspectives from which to investigate the roles of technological artifacts.

By looking through the lens offered by ANT, we examined how dependencies in code changes were managed and what roles technological artifacts played in OSS 2.0. Our analysis described various actors in OSS communities. Among those, we observed that company teams generated boundaries around themselves, which engendered competing networks against the network of coordination across the whole OSS community. Further, we found that artifacts, such as code, were centerpieces of coordination by exercising agency over human actors. Therefore, a significant contribution of this paper is a taxonomy of the various actors and their influence network for coordination across different levels manifested by technological artifacts.

BACKGROUND

Coordination in Open Source Software Development

OSS development has been studied by many scholars (see [1,15] for reviews). OSS communities have been viewed as a unique context to study coordination in distributed work in particular due to the nature of open collaboration [29]. OSS refers to the software release under an “open source” license that allows inspection, modification, and redistribution of the source code of the software [24,44]. Many projects are also open in accepting contributions from any interested developer. 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 infrequently meet, relying on communication tools such as video conferencing, chats, and source code control systems [4,19].

Early OSS developments were driven by voluntary individuals working independently [5,20]. Thus, coordination in OSS has been understood mainly as how distributed individuals manage dependencies [40]. For example, superposition was proposed as a core mechanism of OSS collaboration, by which developers contribute small chunks of code (i.e., atomic commits) to the existing codebase and build on existing functionality [31]. Also, artifacts, such as patch developments and feature requests, were found to facilitate coordination by connecting individuals with similar motivations into cohesive distributed teams [44].

Theory of coordination highlights that distributed developers manage dependencies through sociotechnical affordances of various artifacts [14,15]. For example, documents, including bug reports, provide visibility and accountability of the current state of the work and inform developers of what can be done next [14]. As a result, developers can often coordinate their actions by reference to the code rather than requiring explicit discussion, a process akin to the biological process of *stigmergy* [4], meaning coordination through signs.

However, current research in OSS projects demonstrates that company teams play an increasing role in development in many projects [21,39]. Fitzgerald [24] suggested that the transformation by the company involvement occur across the whole OSS circulation, including development life cycle, product domains, business models, product support, and licensing. Indeed, software companies are shown to reduce development cost by open-sourcing the effort and become more innovative by engaging with OSS communities [2]. Schaarschmidt, Walsh, and von Kortzfleisch [39] described various ways in which firms influence OSS projects. Germonprez et al. [26] found that software designers from software companies can benefit from the engagement with OSS communities for a more productive and creative design for their products. Mäenpää et al. [37] viewed a OSS community as a hybrid system that embed software companies and governments as part of a large software ecosystem.

Despite the increasing involvement of company teams in OSS, research still lacks empirical evidence on how the various actors in the OSS projects manage dependencies to turn the collective effort into a software product [36]. Part of the reason is the complexity in the collaboration among the different actors and stakeholders in OSS communities [36]. The addition of software companies in OSS can change coordination because their motivations and work styles may not align with a given OSS community [2]. Hence, understanding the collaboration network in OSS is often incomplete or does not include important actors [30,31]. Thus, the aim of this paper is to better understand OSS by identifying actors in the scene and mapping them in a complex coordination network using ANT.

Actor-Network Theory

We employed actor-network theory (ANT) as a theoretical framework to describe the pattern of coordination in OSS projects (despite its name, ANT is more of a descriptive method than a theory). Originating from science technology studies (STS), actor-network theory has been a useful lens to examine the relationships and dynamism in sociotechnical systems by illustrating how interests, desires, and agency of different actors interact, compete, and work together (e.g., [3,28,38]).

ANT was originally created to describe the creation of scientific findings, though it has been since extended to study other institutions. The original insight was that scientific findings (and so other institutions) are the product of—and indeed, should be thought of as actually being—networks of "heterogeneous materials": the scientists making the findings, but also lab equipment, supplies, results, papers, etc., all of which must be assembled together to support the findings. Actors themselves can thus be described as networks, and networks can be actors, hence the term actor-networks. From the ANT perspective, successful coordination of software changes can be viewed as building a network that supports the code changes.

In ANT, an *actor* (derived from a semiotic definition of actant) is defined as an entity that acts, that is, that makes a difference in the world, or to which activity granted by others [33]. The ANT approach thus prompts sociotechnical scholars to consider new actors and their behaviors [38,46]. In that an actor can be literally anything if it is the source of an action, actors can include both human and non-human entities [33]. This approach prompts scholars to view technological artifacts as actors, to examine the differences they make to outcomes of interest and to explore how they are brought into ("enrolled into") networks.

In analyzing the interchange among actors, ANT makes a distinction between *intermediaries* and *mediators*. Intermediaries simply transfer meaning and information without altering the actor's behaviors, intentions, and expectations [33]. Intermediaries are thus uninteresting for study. In contrast, mediators "transform, translate, distort, and modify the meaning or the elements they are supposed to carry" [33:39]. A claim of ANT is that more actors should be viewed as mediators rather than intermediaries. For instance, rather than considering communication tools as simple intermediaries, faithfully transferring a user's intentions, we should look for ways in which the tool transforms those intentions.

Building and then maintaining a network is an ongoing activity. A key concept in ANT is *translation*, a process through which diverse actors within an actor-network seek to align others to their interests to bring together an actor-network [8,34]. As Callon and Latour [10] put it: "By translation, we understand all the negotiations, intrigues, calculations, acts of persuasion, and violence, thanks to which an actor or force takes, or causes to be conferred on itself, authority to speak or act on behalf of another actor or force”.

There are four moments of translation discussed in one of the seminal articles that describe the application of ANT to scientific research: Problematization, Interessement, Enrollment, and Mobilization [8]. *Problematization* refers to a process of identifying, defining, and describing actors and their interests and actions that revolve around a focal concern within an actor-network, which is usually labeled as an *obligatory passage point (OPP)*. *Interessement* follows the problematization by illustrating “the group of actions by which an entity attempts to impose and stabilize the other actors [8:207].” *Enrollment* is a process through which the defined actors and their actions in problematization and interessement are accepted, negotiated, and aligned to be included in the actor-network [8]. Callon highlighted that the moments of translation are not permanent and described *mobilization* as a process in which the defined actors and their representativeness can be displaced and modified by other actors as the actor-network faces consistent updates [8].

Some networks may become sufficiently stable that they become taken for granted or black-boxed, in much the same way that the details of infrastructure such as electrical power usually remains unobserved, at least while it is working. This hiding of the details of a network is the process of *punctualization*, by which an actor-network is treated for a while as an individual actor, bracketing the complexity of the whole network with multiple layers of networks [34,38]. Collaborative teams can be an excellent example of punctualization. A team comprises several individual team members as actors, each of whom possesses own interests. However, the team also manifests its own actions as a result of the synthesized actions of the team members. In this case, teams deserve to be viewed as an individual actor that embeds interactions and networks of its members. Punctualization allows researchers not only to reduce the complexity of the multi-layered network but also to view in different perspectives by inviting the punctualized package to the network. However, the simplification should be precarious, as Law argued [34], by paying attention to the individual actors within the punctualized group. It is because not always the group as a network package summarizes and represents all individual actors in it.

Finally, technological artifacts often display properties of the *irreversibility,* meaning eliminating possibilities to go back to possible alternative states once inscribed (e.g., software standards) [7,46]. As Bowker and Star argued [7], software standards, for example, are the inscription of values, opinions, decisions, and interests of developers, users, markets, and policymakers, negotiated during the process of its creation. Thus, software standards are the materialization of social interactions among actors, tend to persist within the network, and thus often are taken for granted by actors.

Only a handful of studies have employed ANT to study the workings of OSS projects. One example is Ducheneaut [18], who illustrated the socialization processes of new participants in an OSS community. Despite this lack of use, the ANT approach is not only suitable but also useful for studying OSS communities for several reasons. First, research has noted the increasing presence of company teams in OSS communities [24,26]. The ANT approach suggests describing the effects of companies on development by suggesting that we consider them as a new actor and examine how their behaviors and agency interact with others. ANT is particularly useful in this effort because it makes no *a priori* assumptions about the size of actors [10] but instead views the size as an outcome. Second, although previous studies have noted the vital role in the open collaboration with technological artifacts, such as source code control systems and communication tools, past research has viewed them as tools that simply convey social cues [16] rather than as actors in their own right, as in ANT.

METHOD

Data Collection

We conducted semi-structured interviews to obtain an in-depth understanding of coordination in OSS communities. Our research team recruited 20 developers from several OSS projects. Because of the exploratory and inductive nature of the study, our goal was to develop a diverse sample of subjects that spanned multiple heterogeneous contexts, rather than a representative sample of a focused population.

The recruitment of the subjects had two phases. In the initial stage of the study (Phase 1), we employed a snowball approach by getting referrals from attendees of relevant conferences. For instance, one of the authors attended ApacheCon[ii] North America in 2017 and was introduced to several contact developers at the venue. Our data includes four subjects (e.g., P14, P15, P16, and P18) recruited at this stage. These initial points of contact were used to develop and validate the interview protocols and to identify major actors in OSS communities.

In the main part of the study (Phase 2), 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 at Phase 1, which included questions regarding subjects' professional background, general descriptions on OSS projects they contributed to, software development tools used for individual work and teamwork, and how they managed dependencies with other developers. Each interview lasted about an hour, either in person or online. Interviews were audio-recorded and then transcribed.

Subjects were diverse regarding tenure in OSS communities, employment type, and participating projects (Table 1). Most of them had more than one year of experience of participating in open source communities. Four subjects have been participating in open source communities as voluntary individual developers, while sixteen subjects have been affiliated with a for-profit company and paid by them for the contribution. Subjects were based in several countries, including Canada, USA, Thailand, South Korea, India, and Germany.

Because subjects expressed concerns regarding the possibility of being identified by peer developers of their open source communities, some projects affiliations had to remain unspecified in this study. Hence, we do not show all subjects' specific affiliations, and in quotations, we use pseudonyms in place of the names of individuals and software companies.

=====Table 1 here====

Data Analysis and Analytical Frame

The transcripts were coded applying an inductive coding approach to identify themes emerging across the subjects [12]. The initial coding was an iterative process, as we tried to identify themes related to actors, their interactions, and networks. Through this process, we discovered themes related to intentions, interests, actions of people, and the roles of technological artifacts. We then employed the open coding to obtain 18 high-level categories and generate a few running hypotheses. Our research team discussed consistently throughout the data analysis. During the theoretical saturation across multiple iterations of coding and categorization, we noticed that several ANT concepts emerged from our data. As we further analyzed our data, we identified distinctive patterns to be categorized into the core concepts of ANT. The actor-network as an outcome of the analysis was repeatedly revised and evolved. The analysis continued until the data were theoretically saturated.

FINDINGS

We present our findings organized by the key concepts of ANT. First, we provide a list of actors and define the obligatory passage point (OPP) as a mutual concern across the actors. Then we present descriptions of their interests as a process of problematization and interessement. Specifically, as an effort of illustrating the interessement, we also discuss how the actors align and negotiate their interests with other actors within the network by showing their struggles to juggle between competing interests as an individual actor and an element of a punctualized group. Then, we discuss the enrollment of the actors through which the non-human actors demonstrate agency and play active roles in attracting and enrolling other actors to the network.

Interdefinition of the Actors and Obligatory Passage Point (OPP)

Various *actors* emerged to play significant roles in open source collaboration and manifested socio-technical issues associated with other actors. As ANT suggests, we did not intentionally isolate technological artifacts from the network but tried to ascribe as much of agency to them as to human actors. We identified four human actors: contributors, committers, company teams, and software companies. These actors are interconnected, as an individual may be affiliated with a company team in an organization. On the other hand, code and software development tools were identified as non-human actors. They emerged as integral part of understanding coordination, in that they delivered signals on what needed to be done and illuminated the power dynamics.

The whole series of actors are participating in a problem by establishing their identities and links between them – building an effective open source software product with few errors and bugs in a community. We call this mutual concern across the actors the obligatory passage point (OPP). Table 2 gives working definitions of each actor we identified and their representative quotes best illustrating their characteristics.

=====Table 2 here====

Description of Actors and Interests: Problematization

Contributors and Committers

Following the definition by Apache Software Foundation (ASF[iii]), a *contributor*[iv] is defined as "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 began as a 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 into 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 actual changes to the shared source code of an OSS project. Their rights regarding the source code involved both writing code and adding changes submitted by contributors to the codebase (i.e., making commits). In our data, there were twelve subjects who were committers in at least one OSS project (see Table 2). 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, and writing code by themselves (P2, P4, and P5). P4 defined a committer: *"I've been a committer, as in someone who can actually change the source code and has control over* *the source code."* As P9 described, committers handle *"virtually every one of the pull requests that go 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 could 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 was common across our data and was more prominent among people who had 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."*

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. A company team in this study should be conceptually distinguished from a network of people collaborating on the same module of an OSS with a perceived team membership but without a common affiliation. Among the twenty subjects, sixteen indicated that they were paid to work on an OSS component of a product of their organization. The other four subjects were not affiliated with a for-profit organization and contributed voluntarily.

Team size varied from small with 3-4 people (e.g., P2, P10) to large with 7-8 people (e.g., P8, P13, P16). Our subjects had responsibilities for a component of a software product of their company, which was closely related or benefited from the collective effort of development in an OSS community. Not all members of a company team were involved in the project to the same degree of time and the amount of code contribution. Our subjects were a point of contact between the software 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 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."*

The work of company teams includes contributing code changes for a software component and providing inputs for contributions from other community developers. The software components that company teams were involved within an open source project were used to develop and enhance commercial products of their company (e.g., P1, P8, P10, P13, and P16). For instance, P16 described the degree to which their team relied on open source software to develop their products: *"Honestly, I hardly use any proprietary technology. We heavily rely on open sources. Starting from the lowest level, we use Linux kernel Digital Ocean, Xen hypervisor, a programming language is Ruby on Rail and open source commercial web server, this means that even though it is an open source, we pay them to unlock some features. And the application layer, we use Ruby Gem, Angular JS, and jQuery plugins."*

Company teams had internal meetings to determine what to do for their product. The internal meetings included having discussions on components to add, scheduling, and reviewing code before submitting to the OSS codebase. Thus, contributions by the company teams were aligned with the software company's technical interests and resources, such as time and human resources (P1, P10, P18). P10 explained the internal coordination 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."*

Contributors or committers who were a member of a company team played role as a contact point between the company team and the open source community. Subjects like P9 were hired to develop a software solution for their organization and collaborated with other developers in an open source community who are in the development of similar solutions for their own organizations. In this case, team meetings involved a developer who contributed to and communicated with an open source community and other team members whose work was less related to developing technical components of the software. 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). P13 described his team and meetings: *"There's a scrum team where I am part of it. It's around eight people in a team, so usually, we go into the meeting in an actual meeting room, or we just do it online."* As P16 emphasized, team meetings played an essential role to reach common grounds among the team members, including both technical developers participating in OSS and non-technical members outside OSS, by ensuring that *"every team member should understand the project from the lowest to the top layer because it is impossible to efficiently work on one layer without knowledge in other layers."*

Software Companies

Software companies were relatively less visible but still appeared to play significant roles behind the scene. Software companies have been viewed as sponsors of OSS projects [35], but in our study, subjects described them as exercising more agency than simply being sponsors in the background. So, we treated them as an entity that possessed their own collective characteristics and intentions. For instance, teams and software companies perceived ownership of a particular module of an OSS project in case the module was 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."* P8 explained how software companies engaged with open source communities: *"My objective is to ensure that we basically—you know all of our effort in the open-source community is aligned to our business needs, we continue to do things that are important to us and we basically also you know engage with some open source community members directly if something [is] required."*

Duality of Developer Identity: Negotiating Interests as Interessement

All but four of our subjects were members of a team affiliated with a software company. Subjects from company teams were deployed to an OSS project by their employer, who was thus a stakeholder in the development community. The individual contributors had duality in identity, such that they participated in the community as an individual, while they represented the interests of their company team and the software company. P8 described: *"I think you know you are part of the open source community, you wear that open-source hat, but still you do represent and 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 code changes and feature implementations were made in 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 software company and the community. P16, as a team leader, emphasized: *"I am a decision maker, so basically I assign a task to each team member based on his/her strengths. They will see their assigned tasks in issue trackers."*

For small changes, the company-affiliated contributors retained agency by bypassing explicit communications within their team. Fixing bugs and typos are good examples of such small contributions. P1 said: *"That's not the right work, but just looking for a sort of 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, committers had more agency to choose what to do and thus made changes to the project based on their greater agency. However, the greater agency of committers was not fully independent but still influenced by their company. Specifically, while contributors' activities were dictated more by their employer's needs with little space for their personal interest, committers were able to exercise more autonomy while considering 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, individuals had multiple memberships in and outside their employer. They had roles acquired based on the amount of code contribution and tenure in the community. The membership of company teams resulted in the duality when exercising their agency as an individual participant of a community. The agency was performed by deciding what to do and how to 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.

Interessement includes any action by an actor to stabilize their identity and to consolidate a system of alliances with other actors. Hence, interessement of an actor is always competitive and relative to other actors in a network. The duality of identity resides between two different associations with developers from company teams: their company team vs. the open source community. From ANT perspective, the duality of identity results in an effort of establishing a system of alliances by developers from company teams to negotiate their interests with other actors, such as their team, the company, and other participants in the community. For contributors and committers from company teams, the effort is on-going and hard to settle during their tenure in a community due to various reasons, such as changing positions in the community (e.g., a contributor becoming a committer), losing or gaining interests in certain parts of code, or the company's strategic decisions for the product.

We interpret the identity duality as a way of maintaining a balance of power throughout the process of interessement. Although the developers from company teams seemed to struggle wearing two hats at the same time, they manifested agency as *mediators* who conveyed influences from their company to an open source community. What was discussed during internal meetings was not always transferred to the community because they had to re-negotiate the interests and demands of other actors in the rest of the community, which included other developers from competitor companies, individual contributors, project visions, and norms within the community. ANT, in this sense, helps us to see the roles of company teams and developers from the teams in constructing a system of alliances to establish an actor-network.

Attraction, Guidance, and Enrollment of Code and Software Development Tools

Code

Code was found to play significant roles in coordination in OSS communities. On the one hand, code was the primary artifact as an outcome of the collective effort in an OSS community. On the other hand, the code acted through its affordance of delivering meaningful signals for coordination, i.e., supporting stigmergic coordination [4]. Stigmergic coordination relies on the shared work outcome to assign labor to workers involved in collaboration without the need for explicit coordination. Code, from this perspective, helps individual developers determine tasks to perform (e.g., which bug to fix or what component is needed) based on their understanding of the source code that has been collaboratively structured in a project community.

Code helped developers who just joined the community get onboard and maintained their participation in an OSS community. 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, and 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 containing histories of the past work and awareness of what was being done. P1 added how code was useful to understand a project: *"It's a very useful product and I think it's very interesting what they're doing with software. ... Let me go take a look. And the first thing I took a look at was, of course, the codebase because that's what a software project in many ways actually is."*

Furthermore, code filled a role in stimulating actions of developers by the way it was written. This made the code a major actor in the OSS communities and one of the 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."* P4 also mentioned his habit of reading code to gain knowledge on the current status of a project: *"So every day I subscribe to the development list, and then I'll look at code on some regular basis. I'll just click, and look at the patch file, it's just very useful to again understand what's going on."*

Second, structures and norms to write code varied by OSS community, and the code enforced contributors to follow the community protocol (P4, P5, and 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 that's 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, selecting issues to solve, and approving code changes 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 greater 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, the flexible nature of the code enabled the collaboration among the developers through 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. P9 mentioned the *"code malleability"* as an inherent nature of code in open source communities, which was constructed and continuously shaped by the members of a community: "*Somebody else has to look at your stuff. It's not just your project. It's everybody's project. If somebody's changing something you wrote, of course, you may have opinions about it, if other people have stronger opinions, oftentimes you just yield.*" P16 summarized the malleable nature of code: *"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."*

Software Development Tools

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

Issue tracker was the core functionality in Jira, where contributors suggested issues (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 provided awareness of other developers' activities: *"I think that's always the best place to start. It is like, look[ing] at what other people have found and see[ing] if there is anything that interests you that you think fits with your skill level or your level of comfortability with the software, and [you] 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 was not adequately 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. As P11 mentioned, GitHub was right at the core of collaboration in open source communities: *"I mean there's no other way to collaborate. I feel GitHub is probably the best, easiest way, like you start a project and then, you can work on it together, see how changes are made."* Smaller and younger projects were adopting GitHub for its advanced git (i.e., a source code version control system) management. GitHub provided 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 provided notification of changes. They were the primary channel between committers and contributors to discuss the submitted changes with other developers. Pull requests allowed developers to comment in line, tag a person, and easily visualize differences in code. P6 described the use of pull requests: *"[Committers] do comment on pull request and yeah, it's [a] 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 happens on pull requests.”*

Pull requests in GitHub appeared to help developers handle discontinuity and support stigmergic coordination. Developers rely on pull requests when someone has to pick up where the work has been left incomplete by the previous developer who is *“only able to work for the first half of the week (because) that’s all that their institution committed them for. (P1)”* In this case, pull requests help developers overcome discontinuity by conveying the context. P1 mentioned: *“Keep it in context with work that you've done, so that people can, for example, people can pick it up and work on it if you're not gonna work on it. Or so they can review it. Or so they can understand how to encrypt user documentation for it for any number of reasons.”*

As we navigate the interests and actions of the actors, code was found to attract developers and inform them of what has been done and what needs to be done, which we call *enrollment*. Code and software development tools enroll developers to the network by providing a problem to work on, information on the history and the structure of the project, and norms and rules of behaviors to write good code and pull requests. In order for developers to be enrolled to the network, they must read the source code, find a problem, come up with an idea to contribute, and request permission from the community to add their work to the codebase. In other words, for developers, negotiating with code and software development tools envelops negotiating with other actors, such as a contributor writing pull requests based on a shared template, a committer determining the code quality and utility, and company teams attempting to influence the project.

Furthermore, we realized that “who has access to code” or “the degree to which an individual can influence the source code” mattered among the actors. For instance, contributors and committers had different levels of rights to apply modifications to the codebase, and thus, company teams made an effort to make their member a committer in an OSS project.

Contributors and committers were battling and negotiating with each other using pull requests to determine what changes should be accepted or rejected. In ANT terms, this means that code and software development tools were the devices of interessement, in that the interrelated roles of actors and their interests were defined by the amount of influence they presented on code and software tools (i.e., whether they can actually change the source code). In this way, code and software development tools possessed more agency to include or exclude a certain actor to the network and created competing networks between the whole OSS community and company teams. P4 described a situation where enrollment of an actor required enrollment through code changes: *"One of my colleagues wrote a fix for the FL parameter to make it more robust, and it's still not been committed despite having had a couple of conversations."*

Coordination in Open Source Software 2.0: An ANT View

The actors identified above constituted an actor-network for coordination in OSS communities (Figure 1). The actor-network illustrates relationships among the human actors, which are mediated by code and software development tools. Company teams themselves were the actors, but at the same time, as punctualized packages, they held its own assemblage embracing individual developers and team members who were outside an OSS community. In the actor-network, company teams and software companies established impermeable boundaries based on corporate affiliations or software components (the solid lines in the figure), whereas OSS communities were intended to be open with *"low barriers of entry* (P4)*"* from outside (the dashed line). Overall, the actor-networks in the OSS communities were found to be collections of heterogeneous actors who exchanged influences to attain mutual interests through coordination.

===Figure 1 here===

The alignments were not always successful and engendered competitions between the actors whose interests were obscured to other actors in OSS 2.0. Company teams put forth their interests, tried to optimize their product, and prioritized the features requested by customers. Thus, company teams often became possessive of their code and even prevented 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."* As a result, coordination in OSS 2.0 was an enrollment process where there was as much effort of keeping others out as bringing actors into the network.

Such broken alignments engendered disjoints in coordination in OSS 2.0. Company teams created competing networks within the whole actor-network that left individual contributors and committers out of the network. P13 lamented: *"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 you can't never see. No discussion, it's like something is done all of a sudden like Ta-da."* P15 explained the disjoints as a failure of sharing social awareness in coordination between company teams: *"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, it's more likely to be a group of people who are physically working together."*

DISCUSSION

ANT was useful to illuminate the coordination across individuals, company teams, and software companies in OSS 2.0. Because ANT does not limit our observations to predefined groups and their roles (e.g., only developers), it allows us the flexibility of describing emerging actors and their relationships [33]. Our findings were discovered mainly 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 software development tools). Table 3 presents definitions of key concepts of ANT and how they were applied to this study.

Human *actors* include contributors, committers, company teams, and software companies. The concept of *punctualization* allows us to understand how company teams can be thought of as individual actors. The punctualized network comprised individual developers hired by a company to work on open source components and their team members who are not visible in the open source community. Further, company teams, as *mediators*, would be an anchor of different actor-networks that competed to enroll code changes, leading to conflicts in coordinating work among them. On the other hand, software development tools and code were identified as non-human actors. They embed actions and competing interests of the human actors. Our data also highlighted that code displayed agency of attracting, informing, and stimulating developers, as part of *enrollment*. Code was seen as a communal and malleable object that manifests the low degree of *irreversibility*.

===Table 3 here===

Implications for Theory and Research

Companies as New Actors in OSS 2.0

We described the roles of software companies, which were relatively unseen in the network [2,36]. Previous literature primarily viewed that OSS projects were driven by a collective effort of independent individuals [24,31]. The presence of companies was viewed as a type of patronage or sponsor in OSS [35,37]. Companies were mostly providing resources, such as human resources, supporting related conferences, and expanding the user base of the product [24]. Moreover, concerns of individual developers from company teams have been paid little attention in previous research. In this study, we showed how company teams influenced coordination in OSS 2.0 by altering the actions and interests of their members and other independent developers.

Putting the company teams and software companies in the network can provide new insights for understanding the recent trends in OSS. Software companies contributed to the projects by deploying individual developers to an OSS community from a work team, and they leveraged the collaboration between their employees and the community. The individuals from company teams have been 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 transparent forum for a communal resolution of a problem and that there was a lack of translucency among company teams. These disagreements sometimes seemed to hinder coordination among company teams and made it difficult to work together.

By considering software companies as actors, our understanding of other roles in OSS can change. For instance, 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. This suggests that we may have to reconsider the agency of individual participants in OSS development, in conjunction with the growing agency of corporate stakeholders. Specifically, individual developers can still voluntarily exert effort and decide what to do, but at the same time they should negotiate their interests with those of company teams.

Moreover, putting software companies in the network informs our understanding of the governance and control of OSS in the future. Most OSS communities, especially in Apache Foundation, adopt “the consensus-based, community-driven governance” (a.k.a., The Apache Way). However, several OSS communities begin to demonstrate characteristics of company-driven governance. For instance, given that committers possess more agency and power to determine what issues to solve and what changes to merge, software companies would want their employees to become committers [39]. We found that committer positions were deemed as a scarce but powerful resource for company teams because the position could render an opportunity for a company to steer the OSS project. This suggests that the increasing presence of company teams means their greater responsibility in coordination with other actors and the greater power in the authoritative structure in OSS projects [17,40]. Thus, this study not only echoes the notion of OSS 2.0 by Fitzgerald [24] but also puts more emphasis on the agency of companies in OSS governance and control.

The duality of identity contributes to our understanding of what drives individual developers from company teams to maintain their participation, by showing their attempts to achieve a stable identity as a contributor or a committer. The duality of identity was primarily created and reinforced by companies governing individual developers’ agency and autonomy in the form of employment. Existing theories regarding OSS collaboration has mostly treated individual developers as those who have clear motivations to participate, which can be either intrinsic or extrinsic [42,45]. However, our findings can open an interesting avenue to consider the competing interests of one individual contributing to multiple groups. Construction of a stable identity has been found to sustain the participation of individual developers [20]. Future research should explore which identity triumphs in a specific circumstance of open source projects. One approach for this question is to examine the duality of identity and developer behaviors based on a typology of OSS projects. For instance, Schaarschmidt and colleagues [39] categorized commercializing approaches of OSS projects into four types based on initiating party (firm vs. community) and the number of the participating firm. Although it appears that the duality of identity is observed across OSS projects in varying sizes in our data, a typological approach can help provide a more nuanced understanding of the phenomenon.

Besides, from an organizational perspective, an essential empirical concern is whether the companies as new actors actually enhance the quality of open collaboration and the longevity of the project [11]. Companies may bring more resources and workforces as an investment to an OSS project. This is, in part, the reason why the increasing presence of companies in open sources scenes are often positive [2,26]. However, a similar trend is viewed worrisome in other domains that rely on open collaboration. Companies and even resourceful individuals not only provide financial support but also make attempts to edit Wikipedia entries to their advantage [27]. Our findings based on ANT can extend to similar domains benefitting from corporate support, such as Wikipedia and citizen science, to better understand the roles and interests of various actors around the community boundaries and their influences on the project quality.

Redefining Roles of Code in OSS 2.0

Analyzing OSS projects through the lens of ANT prompted us to attribute more agency to non-human actors. Code turned 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 in changing behaviors of individual developers, company teams, software companies, and eventually, the direction of the project development throughout the community. 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.

Also, software development 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. The technical affordance of the software development tools, including notifications and commenting in line, provided social awareness and contextual continuity to developers working in different time zones and locations.

Theoretically, ANT offers a new angle to view code in software development. Previously, software has been viewed as a *frozen* material that *hides* an organization's values and actions [7]. This view toward code has been adopted to studies on sociotechnical systems, such that code materializes properties of irreversibility in actor-networks [46]. However, we showed that the codebase, as a flexible, collective organism, is at the core mechanism of coordination in OSS projects. The findings suggest that we may need to treat code as a shared material that possesses low degrees of immutability and instead embeds multiple alternative ways of constructing the software and the history of changes over time.

Our findings on the flexible and informative nature of code can inform stigmergic coordination in OSS projects. As observed in social insects, such as ants and honey bees, constructing a collaborative outcome (e.g., building a colony or a nest) without explicit coordination, stigmergic coordination utilizes a shared material itself as a coordination mechanism without additional articulation work [4,14]. In the context of OSS collaboration, the codebase contains information regarding what has been changed, what is done already, what is in progress, and what needs to be done. Therefore, the code can inscribe the traces of dependencies among open source participants. Previous works on open source software (e.g., [16]) mainly focused on the technical affordance of collaboration tools and treated the code as a work outcome. We provided evidence that code contributed to stigmergic coordination by externalizing and conveying the history and the knowledge on the system and other actors' behaviors. Future research should examine the role of code and other shared work outcomes in similar settings and explore boundary conditions that enable stigmergic coordination.

In addition, understanding how different actors react and behave related to code should be an interesting theoretical concern. Previous studies on OSS demonstrated that the success of open source collaboration depends on trust among developers, which comes from transparent process beliefs and a shared vision on the project [43]. Our data showed that committers and contributors had different attitudes and practices in writing and interpreting code. The committers were 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). If the differences in coding practices engender tensions among contributors and committers and between company teams, the shared codebase and supporting tools like pull requests can mitigate the tensions by offering translucency of code changes (e.g., what is removed and added along with who made the change) and thus promote trust among developers from different background across the OSS community.

Implications for Open Source Software Development

This study provides practical implications for entities participating in open source projects and companies interested in leveraging the open source development for their business.

Although the corporate involvement in open source scenes has been observed for a while, the type of engagement and how organizations utilize open source communities have recently been changing. One of the most common ways for companies to leverage an open source approach was to release a software component to open source communities and invite developers to engage and enhance the component [25]. OpenOffice software suite, for example, had been set out to be open source by Sun Microsystems and improvements in the suites were reflected to Sun's proprietary office applications[v] [46,47]. Walmart adopted a similar model: OneOps, a cloud management tool, was open sourced in hopes to bring open source developers into the enhancement of the software [22].

However, we observed that companies went beyond attracting individual developers and deployed their employees to open source communities. The company involvement becomes prevalent in projects besides ones mentioned in this study, including MySQL and RStudio. Most contributions for MySQL are by people from companies, and they become a significant force to move the community forward. The influence of open sourcing has been impactful enough to attract individual developers. Developers dive into open source projects to increase their visibility within the open source repositories, which could potentially improve one's reputation and hence, enable enhanced career opportunities.

The increasing company involvement poses a critical dilemma for the future of open source software: open licenses vs. open collaboration. There has been a long belief that software applications with open licenses are made possible by open collaboration of benevolent, voluntary individuals [20,30]. Thus, the open collaboration of developers is well aligned not only with the ideology (e.g., "open source way") but also with how the software product evolves and improves. In our study, the Actor-Network of open source communities revealed impermeability between the open source boundaries and company team boundaries, through which code changes and intentions were not openly shared. The findings point to the fact that open source projects with the dominant company involvement may be deprived of the benefits of open collaboration in the long run, such as consistent evolution, community support, and no vendor lock-in.

As a result, companies and IT administrators will face a new challenge to align their strategies and business interests with other companies in an OSS project. The alignment with other companies can be as important as those with the whole project community when the project is actively driven by company teams. In this case, team members who interact with open source communities can be a passage to such alignments. As a way to increase the software's use and usefulness, companies may wish to foster effective correspondence with other company teams as well as the community as a whole.

Also, it will be imperative for companies and project administrators to mitigate the negative ramifications of the duality in their developer's identity. For instance, companies can devise a way to evaluate a company team's performance by the extent to which the team is responsive to address issues related to the whole open source project as well as a particular component only for the company. Given that the company team's engagement with a community can determine the quality of the company's product, it is crucial for companies to effectively manage the competing interests of the company teams, which are split between the whole OSS community and the company product. In situations where a company is trying to ensure adoption and usage of their software, it is possible that being a bridge to the external community can be a key enabler of the software's use and usefulness. We believe that, in this way, company-affiliated developers can harness the duality of identity by being genuinely motivated to contribute to the project following their own curiosity as an individual developer and, at the same time, by being properly incentivized by their employer.

Code was viewed with agency and capable of delivering various signals for coordination. Thus, tools for writing and submitting code should provide more nuanced social awareness. There were certain norms and rules to write good code to be reviewed by committers. The software development tools can embed a community-approved template including such information as the writer, the logic behind, and their affiliation.

Finally, software development tools should support modularity in the codebase. Our findings suggest that companies work on different components of the source code, but the boundaries among the components were not clearly negotiated nor defined. Software development tools can improve coordination among the company teams, for example, by showing specific lines that are affected by new commits and providing enhanced searchability. In those ways, work can be shared without version conflicts and avoid duplication of effort.

Limitations and Future Research

First, our data suffer from a limitation that our subjects may not represent the whole OSS community. We admit that the actors' roles and interests are dynamic, rather than persistent, and so is the actor-network that consists of the actors. Thus, the last moment of translation should be to acknowledge the possibility of *betrayal* of the actors who can change their interests at any time and update the actor-network with *mobilization* of the actors beyond this study [8].

By having subjects from a wide range of OSS projects, we obtained a broad understanding of the trends of company involvement and coordination among them, which seemed to happen across OSS communities in varying sizes. However, we were not able to examine the evolution of actor-networks. It may require other types of data, such as a longitudinal observation. Future studies can benefit from an in-depth case study with one OSS community. The case study method has been applied to describe the process of an organizational change that is enabled by information technology [38,46]. A systematic analysis of the source code, accompanying the interpretative analysis of developer interactions, can unfold the process of how an OSS product embeds the interactions among the actors over time.

Finally, we interviewed only developers whose contributions were made mainly by constructing code. As we analyzed data, we realized that customers of an OSS product might play significant roles as well. Subjects mentioned that directions of the software and changes in future iterations could depend on inputs from outside the developer communities (e.g., P8, P13). Future studies should expand the actor-network to the whole eco-system of OSS by including actors who are less involved in the code construction.

CONCLUSION

We looked at the coordination in OSS communities through the lens of ANT. We discovered several kinds of human and non-human actors. Software companies were identified as a significant actor in the scene, which has been treated tangentially in OSS communities. We also highlighted the active roles of software development tools and code as mediators, through which norms and information on the work were delivered. Our results inform the understanding of coordination in OSS 2.0 and suggest theoretical and practical implications for OSS.

---

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

[ii] "ApacheCon is the place to come learn what Apache projects are doing, as well as a place for projects to come build stronger project communities, and forge bonds between projects (www.apachecon.com)."

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

[iv] See https://en.wikipedia.org/ wiki/Committer. Contributors are also referred to as developers in some cases (e.g., by the 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.

[v] OpenOffice was discontinued by Sun in 2011 after the project was donated to the Apache Foundation and became Apache OpenOffice. When the project was active, Sun Microsystems managed the project under the Sun Industry Standards Source License (SISSL), a currently retired free and open source license supported by the company. Sun Microsystems was acquired by Oracle Corporation in 2010.

REFERENCES

1. Aksulu, A. and Wade, M.R. A comprehensive review and synthesis of open source research. *Journal of the Association for Information Systems*, *11*, 11 (2010), 576–656.
2. Andersen-Gott, M., Ghinea, G., and Bygstad, B. Why do commercial companies contribute to open source software? *International Journal of Information Management*, *32*, 2 (2012), 106–117.
3. Atkinson, C.J. The "Soft Information Systems and Technologies Methodology"(SISTeM): an actor network contingency approach to integrated development. *European Journal of Information Systems*, *9*, 2 (2000), 104–123.
4. Bolici, F., Howison, J., and Crowston, K. Stigmergic coordination in FLOSS development teams: Integrating explicit and implicit mechanisms. *Cognitive Systems Research*, *38*, (2016), 14–22.
5. Bonaccorsi, A. and Rossi Lamastra, C. Altruistic individuals, selfish firms? The structure of motivation in Open Source software. *The Structure of Motivation in Open Source Software*, (2003).
6. Bonner, W. History and IS–Broadening our view and understanding: Actor–Network Theory as a methodology. *Journal of Information Technology*, *28*, 2 (2013), 111–123.
7. Bowker, G.C. and Star, S.L. Knowledge and information in international information management: Problems of classification and coding. In L. Bud-Frierman, ed., *Information Acumen: The Understanding and Use of Knowledge in Modern Business*. Routledge, London, 1994, pp. 187–213.
8. Callon, M. Some Elements of a Sociology of Translation: Domestication of the Scallops and the Fishermen of St Brieuc Bay. *The Sociological Review*, *32*, 1_suppl (May 1984), 196–233.
9. Callon, M. Techno-economic Networks and Irreversibility: *The Sociological Review*, (May 2014).
10. Callon, M. and Latour, B. Unscrewing the big Leviathan: how actors macro-structure reality and how sociologists help them to do so. *Advances in social theory and methodology: Toward an integration of micro-and macro-sociologies*, *1*, (1981).
11. Capra, E., Francalanci, C., Merlo, F., and Rossi-Lamastra, C. Firms' involvement in Open Source projects: A trade-off between software structural quality and popularity. *Journal of Systems and Software*, *84*, 1 (January 2011), 144–161.
12. Charmaz, K. and Belgrave, L.L. Grounded theory. *The Blackwell Encyclopedia of Sociology*, (2007).
13. Crowston, K. A coordination theory approach to organizational process design. *Organization Science*, *8*, 2 (1997), 157–175.
14. Crowston, K., Østerlund, C., Howison, J., and Bolici, F. Work Features to Support Stigmergic Coordination in Distributed Teams. In *Academy of Management Proceedings*. Academy of Management, 2017, pp. 14409.
15. Crowston, K., Wei, K., Howison, J., and Wiggins, A. Free/libre Open Source Software development: What we know and what we do not know. *ACM Computing Surveys*, *44*, 2 (2012), 7:1–7:35.
16. Dabbish, L., Stuart, C., Tsay, J., and Herbsleb, J. Social coding in GitHub: Transparency and collaboration in an open software repository. In *ACM Conference on Computer Supported Cooperative Work (CSCW)*. ACM, 2012, pp. 1277–1286.

17. Di Tullio, D. and Staples, D.S. The governance and control of open source software projects. *Journal of Management Information Systems*, *30*, 3 (2013), 49–80.
18. Ducheneaut, N. Socialization in an open source software community: A socio-technical analysis. *Computer Supported Cooperative Work (CSCW)*, *14*, 4 (2005), 323–368.
19. Espinosa, J.A., Slaughter, S.A., Kraut, R., and Herbsleb, J.D. Team knowledge and coordination in geographically distributed software development. *Journal of Management Information Systems*, *24*, 1 (2007), 135–169.
20. Fang, Y. and Neufeld, D. Understanding sustained participation in open source software projects. *Journal of Management Information Systems*, *25*, 4 (2009), 9–50.
21. Feller, J., Finnegan, P., Fitzgerald, B., and Hayes, J. From Peer Production to Productization: A Study of Socially Enabled Business Exchanges in Open Source Service Networks. *Information Systems Research*, *19*, 4 (December 2008), 475–493.
22. Finley, K. Walmart Doesn't Want You to Get Locked Into Amazon's Cloud. *Wired*, 2016. https://www.wired.com/2016/01/walmart-doesnt-want-you-to-get-locked-into-amazons-cloud/.
23. Finley, K. Why 2018 Was a Breakout Year for Open Source Deals. *Wired*, 2018. https://www.wired.com/story/why-2018-breakout-year-open-source-deals/.
24. Fitzgerald, B. The transformation of open source software. *MIS Quarterly*, (2006), 587–598.
25. Fitzgerald, B., Kesan, J.P., Russo, B., Shaikh, M., and Succi, G. *Adopting open source software: A practical guide*. MIT Press, 2011.
26. Germonprez, M., Kendall, J.E., Kendall, K.E., Mathiassen, L., Young, B., and Warner, B. A theory of responsive design: A field study of corporate engagement with open source communities. *Information Systems Research*, *28*, 1 (2016), 64–83.
27. Hafner, K. Seeing corporate fingerprints in Wikipedia edits. *The New York Times*, *19*, (2007).
28. Heeks, R. and Stanforth, C. Understanding e-Government project trajectories from an actor-network perspective. *European Journal of Information Systems*, *16*, 2 (2007), 165–177.
29. Herbsleb, J.D. and Grinter, R.E. Splitting the organization and integrating the code: Conway's law revisited. In *Proceedings of the 21st international conference on Software engineering*. ACM, 1999, pp. 85–95.
30. Hippel, E. von and Krogh, G. von. Open Source Software and the "Private-Collective" Innovation Model: Issues for Organization Science. *Organization Science*, *14*, 2 (2003), 209–223.
31. Howison, J. and Crowston, K. Collaboration through open superposition: A theory of the open source way. *MIS Quarterly*, *38*, 1 (2014), 29–50.
32. Latour, B. *Science in action: How to follow scientists and engineers through society*. Harvard university press, 1987.
33. Latour, B. *Reassembling the Social: An Introduction to Actor-Network-Theory*. Oxford University Press, 2005.
34. Law, J. Notes on the theory of the actor-network: Ordering, strategy, and heterogeneity. *Systems Practice*, *5*, 4 (August 1992), 379–393.
35. Lindman, J. and Hammouda, I. Support mechanisms provided by FLOSS foundations and other entities. *Journal of Internet Services and Applications*, *9*, (February 2018), 8.
36. Link, G.J. and Jeske, D. Understanding Organization and Open Source Community Relations through the Attraction-Selection-Attrition Model. In *Proceedings of the 13th International Symposium on Open Collaboration*. ACM, 2017, pp. 17.

37. Mäenpää, H., Fagerholm, F., Munezero, M., Kilamo, T., and Mikkonen, T.J. Entering an ecosystem: The hybrid OSS landscape from a developer perspective. In *CEUR Workshop Proceedings*. 2017.
38. Sarker, S., Sarker, S., and Sidorova, A. Understanding business process change failure: An actor-network perspective. *Journal of management information systems*, *23*, 1 (2006), 51–86.
39. Schaarschmidt, M., Walsh, G., and von Kortzfleisch, H.F.O. How do firms influence open source software communities? A framework and empirical analysis of different governance modes. *Information and Organization*, *25*, 2 (April 2015), 99–114.
40. Shaikh, M. and Henfridsson, O. Governing open source software through coordination processes. *Information and Organization*, *27*, 2 (2017), 116–135.
41. Shaikh, M. and Vaast, E. Folding and unfolding: Balancing openness and transparency in open source communities. *Information Systems Research*, *27*, 4 (2016), 813–833.
42. Singh, P.V. and Tan, Y. Developer Heterogeneity and Formation of Communication Networks in Open Source Software Projects. *Journal of Management Information Systems*, *27*, 3 (December 2010), 179–210.
43. Stewart, K.J. and Gosain, S. The Impact of Ideology on Effectiveness in Open Source Software Development Teams. *MIS Quarterly*, *30*, 2 (2006), 291–314.
44. Temizkan, O. and Kumar, R.L. Exploitation and exploration networks in open source software development: An artifact-level analysis. *Journal of Management Information Systems*, *32*, 1 (2015), 116–150.
45. Von Krogh, G., Haefliger, S., Spaeth, S., and Wallin, M.W. Carrots and rainbows: Motivation and social practice in open source software development. *MIS Quarterly*, (2012), 649–676.
46. Walsham, G. and Sahay, S. GIS for District-Level Administration in India: Problems and Opportunities. *MIS Quarterly*, *23*, 1 (1999), 39–65.
47. OpenOffice.org. *Wikipedia*, 2019. https://en.wikipedia.org/w/index.php?title=OpenOffice.org&oldid=892984881.

Number	Medium	Years in OSS	Involvement	Projects	Country	Roles in Projects
P1	Phone/Skype (Phase 2)	15 years	Full-time Employee	Apache (Jena, Stanbol, Clarezza, Spark), Fedora Commons, and Islandora CLAW	Canada	Contributor and committer
P2	Phone/Skype (Phase 2)	3 years	Full-time Employee	Apache Hadoop (HDFS component)	USA	Committer
P3	Phone/Skype (Phase 2)	4 years	Full-time Employee	Islandora and Fedora	Canada	Contributor and committer
P4	Phone/Skype (Phase 2)	12 years	Full-time Employee	Apache projects and open source Java code quality projects	USA	Contributor and committer
P5	Phone/Skype (Phase 2)	3 years	Full-time Employee	Apache Spark, Lucene, and more (unspecified)	South Korea	Committer
P6	Phone/Skype (Phase 2)	6 years	Full-time Employee	Lucene, Luke, Solr, and multiple other projects	USA	Contributor
P7	Phone/Skype (Phase 2)	5 years	Full-time Employee	Pandas, Jupyter environment kernels, and Pypandoc	Germany	Contributor
P8	Phone/Skype (Phase 2)	7 years	Full-time Employee	Hadoop, HBase, ZooKeeper, BookKeeper, and Hive	India	Contributor
P9	Phone/Skype (Phase 2)	12 years	Full-time Employee	Open ONI, Drupal, and Samvera	USA	Committer
P10	Phone/Skype (Phase 2)	1 year	Full-time Employee	TensorFlow	USA	Contributor
P11	Phone/Skype (Phase 2)	6 years	Voluntary as Individual	Riot.js, Jupyter notebook extensions, TensorFlow, and small unspecified projects	USA	Committer
P12	Phone/Skype (Phase 2)	8 years	Full-time Employee	Apache Spark, Terraform Provider, Homebrew-Cask, and many other	USA	Contributor and committer
P13	Phone/Skype (Phase 2)	2 years	Full-time Employee	vSphere	USA	Committer
P14	Face-to-Face (Phase 1)	8 years	Full-time Employee	Ruby Library and AngelList APIs	USA	Committer
P15	Face-to-Face (Phase 1)	5 years	Full-time Employee	jqGrid and jQuery	USA	Contributor
P16	Face-to-Face (Phase 1)	5 years	Full-time Employee	Linux kernel Digital Ocean, Xen hypervisor, Ruby Gem, Angular JS, and jQuery plugins	Thailand	Committer
P17	Face-to-Face (Phase 2)	1 year	Voluntary as Individual	Several small open source projects through GitHub, and open data projects (not specified)	USA	Contributor
P18	Face-to-Face (Phase 1)	19 years	Full-time Employee	Apache Directory and Apache Mina	USA	PMC, Committer
P19	Phone/Skype (Phase 2)	3 years	Voluntary as Individual	Multiple Open Source NLP tools (not specified)	South Korea	Contributor
P20	Face-to-Face (Phase 2)	2 months	Voluntary as Individual	Various meme generators, a small, open source iOS theming platform (not specified)	USA	Contributor

Table 1 List of Study Subjects

Type	Actor	Definition in This Study	Representative Quote
Human Actors	Contributors	Individual developers who contribute to an OSS community in any form including code, documentation, tests, and ideas in varying degrees	*"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." (P20)*
	Committers	Individual developers who have permissions to make changes to the shared source code of an open source project	*"I've been a committer, as in someone who can actually change the source code and has control over the source code." (P4)*
	Company Teams	A group of people in a company who work together on the same module of a software product	*"There's a scrum team where I am part of. It's around eight people in a team, so usually, we go into the meeting in an actual meeting room, or we just do it online." (P13)*
	Software Companies	A for-profit entity that utilizes the shared outcome of an OSS community in exchange of deploying their resources to the community	*"My objective is to ensure that we basically—you know all of our effort in the open-source community is aligned to our business needs, we continue to do things that are important to us and we basically also you know engaging with some open source community members directly if something required." (P8)*
Non-Human Actors	Code	The primary artifact of a software product as an outcome of the collective effort by members of an open source community	*"It's a very useful product, and I think it's very interesting what they're doing with software. ... Let me go take a look. And the first thing I took a look at was, of course, the codebase because that's what a software project in many ways actually is." (P1)*
	Software Development Tools	A piece of software that members of an open source community utilize to carry out various actions such as discussing and writing code	*"I mean there's no other way to collaborate. I feel GitHub is probably the best, easiest way, like you start a project and then, you can work on it together, see how changes are made." (P11)*

Table 2 List of Actors Identified in Open Source Communities

Concept	Definition	Application to this Study
Actor	"A*ny thing* that does modify a state of affairs by making a difference" [33:71]	Contributors, committers, company teams, software companies, code, and software development tools
Intermediaries	Transfer meaning and information without altering the actor's behaviors, intentions, and expectations [33]	Uninterested in this study
Mediators	"Transform, translate, distort, and modify the meaning or the elements they are supposed to carry" [33:39]	Developers from company teams and code influencing the open source project
Translation	"the interpretation given by the fact or technology builders of their own interests and those of the actants they seek to enroll in order to transform their claim to a matter of fact" [32:180]	See the moments of translation below.
Problematization	"Determining a set of actors and defining their identities in such a way as to establish themselves an obligatory passage point in the network of relationships they were building" [8:204]	Providing interdefinitions of the actors and their interests.
Obligatory Passage Point (OPP)	An event or situation that concerns all the actors and their alliances, and where relationships are formed around [8,38]	An open source software product that operates effectively without errors and bugs
Interessement	"the group of actions by which an entity attempts to impose and stabilize the other actors it defines through its problematization" [8:207]	The duality of identity of contributors and committers from company teams is part of ongoing efforts of negotiating their interests with the actor-network.
Enrollment	As part of translation, a set of defined actors accept, negotiate, and align their interests to be included in the actor-network [8]	Code as an enrollment device as trying to attract developers to work on problems and providing information on the history of work by an OSS community
Punctualization	Reducing a heterogeneous, complex network into a smaller, simple package of actors by identifying network patterns	Networks of individual developers hired by a company and their team members were punctualized into company teams to be individual actors
Irreversibility	"The degree to which it is subsequently impossible to go back to a point where alternative possibilities exist" [38:56,46:42]	The source code is malleable and evolving through a constant collaborative (re)construction by various actors in the actor-network.
Actor-Network	"Heterogeneous network of aligned interests, including people, organizations and standards" [38:56,46:42]	Actor-Network in OSS 2.0 in Figure 1
Note: The definitions in this table were adopted from various sources of theoretical discussions on ANT [9,10,33,34,38,46].		

Table 3 Key Concepts of ANT Used in This Study

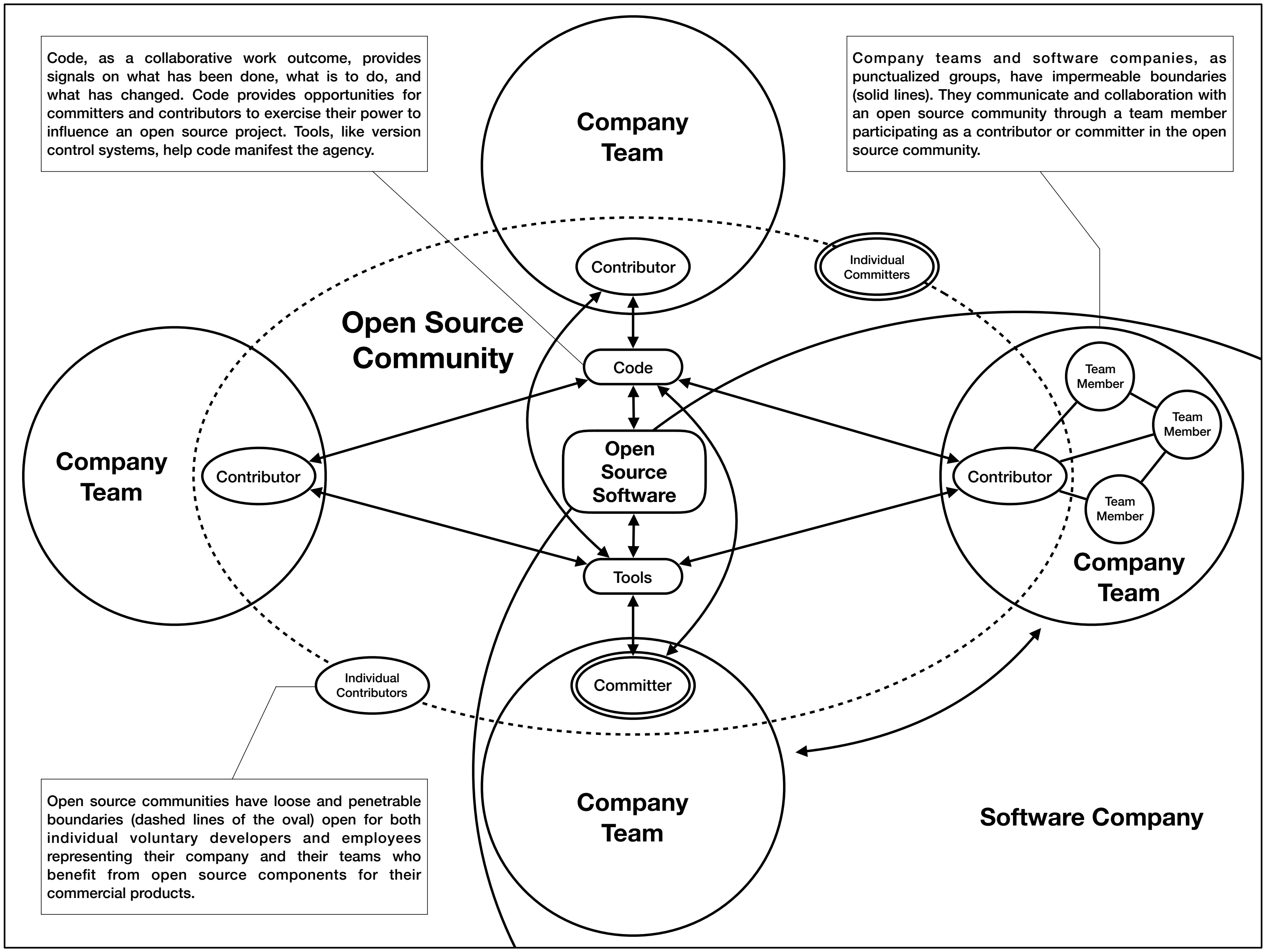
Code, as a collaborative work outcome, provides signals on what has been done, what is to do, and what has changed. Code provides opportunities for committers and contributors to exercise their power to influence an open source project. Tools, like version control systems, help code manifest the agency.
Company teams and software companies, as punctualized groups, have impermeable boundaries (solid lines). They communicate and collaboration with an open source community through a team member participating as a contributor or committer in the open source community.
Open source communities have loose and penetrable boundaries (dashed lines of the oval) open for both individual voluntary developers and employees representing their company and their teams who benefit from open source components for their commercial products.
Company Team
Contributor
Individual Committers
Open Source Community
Code
Company Team
Contributor
Open Source Software
Team Member
Team Member
Contributor
Team Member
Company Team
Tools
Individual Contributors
Committer
Company Team
Software Company

Figure 1 Actor-Network in OSS 2.0