Theorizing Modes of Open Source Software Development

Aron Lindberg
Case Western Reserve University
aron.lindberg@case.edu

Xuan Xiao
Harbin Institute of Technology
xiaoxuanhit@gmail.com

Kalle Lyytinen
Case Western Reserve University
kjl13@case.edu

Abstract

Open Source Software (OSS) development is distributed across actors and artifacts and involves translating diffuse representations into distinct sets of contiguous code artifacts. Despite the highly distributed and dynamic nature of OSS development, it is often described in unitary, monolithic terms – an unfortunate situation which masks considerable variance across OSS development processes. Therefore we explore reasons for systematic variance in these processes so as to enable more effective OSS development practices. Drawing on theory of distributed cognition, we develop a language of cognitive translations, which occur within and across distributed social arrangements and structural conditions of sharing knowledge. This language provides micro-foundations for understanding how different modes of OSS development emerge. Through examining how generative characteristics of social and structural distributions in OSS shape distinct development pathways, we propose a theoretically derived typology explaining the characteristics, dynamics, and conditions for success of different modes of OSS development.

1. Introduction

OSS is increasingly becoming an important way of organizing software development processes. However, extant accounts often conceptualize OSS development as a unitary, almost monolithic, mode of organizing [20]. This is problematic because assuming a single modus operandi results in simplistic understanding of OSS development. Consequently, considerable variance in OSS development processes is masked and hidden away. Even though some research [19,24] has been conducted that recognizes the diversity of OSS organizing modes, they do not provide plausible explanations of underlying mechanisms of how and why different modes emerge.

In order to explain different modes of OSS development, we propose that it is important to analyze cognitive translation processes, which translate ambiguous and distributed representations of OSS requirements into definitive sets of artifacts expressed in working code. Such translation processes are firmly grounded in what we call the *micro-foundations of translation* referring to distinct modes of cognitive transference and transformation across social and material actors. Understanding how such micro-translations are configured differently across contexts can help us distinguish between different modes of open source organizing. In short, they show how cognitive activities associated with OSS distribute and configure themselves across time and space.

In order to understand the distinct ways in which translation processes are configured, we explore two dimensions - social distribution and structural distribution. These distributions form the underlying forces that shape the emergence of particular sets of translations. Through examining how OSS translation processes unfold differently as a function of their underlying social and structural distribution, we can identify distinct modes of OSS development and recognize the specific conditions for success that each distinct mode of OSS development entails.

The paper is organized as follows: we first explore the micro-foundations of OSS development through formulating a sociotechnical language for describing distributed cognitive interactions across social and structural distributions. We contend that these micro-foundations consist of four types of translation - types of transferring and transforming cognitive representations across social and structural distributions: 1) communicative-, 2) inscriptive-, 3) retrieval-, and 4) automatic translation. Subsequently, we conceptualize a typology of related OSS development modes: four typical ways in which variations in social and structural distributions lead to distinct configurations of translation processes so as to create working code. We conclude with discussing the implications of our conceptual framework as well as future research directions.

2. OSS Development

In this section, we review past research on OSS, focusing especially on the emerging understanding of variation across different forms of OSS development. Further, we approach OSS as a distributed sociotechnical cognitive system, which involves social actors voluntarily participating in OSS development and the material artifacts with which social actors interact. The cognitive distribution of OSS development can be characterized by describing the underlying social and structural distributions. Hence, distinct combinations of distributions configure knowledge translation processes differently so as to enable requirements to be iteratively translated into code along distinct and varying pathways.

2.1. Past Research on OSS Development

In recent years, OSS has attracted an increasing number of researchers to explore its various aspects with respect to its development, such as motivations [1,30], organizing and governance structures [6,10,19,23,31], and evolution patterns [8,15]. Some distinguishing features of OSS development include voluntary developers [6,30] and use of artifactual 'informalisms' [27,29]. These features illustrate the distributed and emergent nature of OSS projects.

The early groundbreaking and canonical work on the OSS development employs the metaphor of 'bazaar' [24] to characterize the organizing type of OSS development, which involves a great amount of anonymous developers and users with distinct expertise and viewpoints devoting to source code. However, more recent accounts such as [16] and [19] portray a different reality. These accounts provide rich descriptions of the actual modus operandi of OSS development by empirically examining various OSS projects. Their findings suggest that OSS management and organizing do not necessarily resemble a Raymondian 'bazaar'. Rather, many OSS projects are driven by a core team composed of several highly committed developers, often sponsored by corporate entities, and assisted by a growing periphery of developers and users who report bugs, suggest bug fixes and, most importantly use the software and provide feedback [5].

Hence, the contemporary OSS literature is increasingly recognizing different ways in which OSS development processes vary. For example, it has been shown that the ways in which codebases change follow multiple patterns such as linear [25], superlinear [8] and punctuated equilibria [3]. Further, the organization of developer communities also demonstrates variation around the generic core-periphery model proposed by Mockus and his colleagues [19].

Taken together, there seems to be ample evidence that the ways in which the social and structural dimensions of OSS projects are arranged exhibit considerable variance, and therefore we can expect that the ways in which OSS development processes unfold under these conditions also show similar degrees of variance.

2.2. Social and Structural Distribution

As informed by the previous research, variances of OSS development processes are highly associated with varying configurations of and interactions between actors participating in OSS development as well as the artifacts they use. Therefore, we draw on theory of distributed cognition [12] and approach OSS development as a process through which distributed sets of actors and artifacts work in concert to translate abstract representations into tangible, working code. This lens helps to recognize not only the presence of actors and artifacts, but also the processes through which they interact to develop OSS.

The theory of distributed cognition explains how a feasible solution to a collaborative task emerges from computation of representational states over time [12]. The representational states could be internal representations residing in individual minds or external representations embodied in artifacts and physical environment [22,26]. Consequently, cognition is distributed across social and structural boundaries with cognitive workload shouldered upon actors and artifacts involved in a specific task [11,13]. To capture these forms of distribution and the dynamics between them, we identify two such forms: social- (i.e., the distribution of cognition among actors), and structural (i.e., the distribution of cognition across artifacts) distribution [14]. Both of these distributions play vital roles in understanding how collaborative tasks, such as OSS development, are carried out.

In OSS development, *social distribution* indicates that cognitive workload is distributed across multiple and varied participants of an OSS community. It has been observed that OSS development is carried out through diverse volunteers playing specific roles [5,33] with significant diversity of knowledge [21,28]. For instance, Crowston and Howison [5] claim that OSS communities are hierarchical, onion-like organizations. At the center of the onion, a core of developers takes charge of the overall design. As developers move away from the center towards the outer layers of the onion, the engagement level tends to decrease [5, 32]. Interactions of multiple actors who devote diverse cognitive resources and technical expertise to write code, report bugs, and give suggestions provide an important impetus for OSS development.

Consequently, the social distribution denotes the complete set of human actors involved with a specific OSS project.

Structural distribution captures how cognitive workloads are supported by sets of artifacts that the OSS community uses. Hence, the structural distribution identifies the complete set of artifacts involved with a specific OSS project. Given the dearth of formal control in OSS, development depends on ad hoc artifacts, often labeled 'informalisms' [29], such as mobilized webpages (e.g., project Wiki pages, Readme profiles), various tools users work with (e.g., Internet Relay Chat, project mailing lists, or project testing suites) and bug tracker/code hosting platforms (e.g., GitHub, SourceForge), which serve as archives for OSS development. It is these heterogeneous forms of artifacts and their affordances that help shoulder and disseminate cognitive workload in OSS projects [27,29] so as to push OSS development processes forward.

3. Micro-Foundations of OSS: Translation Processes

The forms of interactions within and across social and structural distributions of OSS development communities lead to a number of distinct translation processes that work towards computing requirements through collating and instantiating distributed and abstracted requirements into distinct sets of contiguous software artifacts. Translation processes exhibit the ways in which software related knowledge is transferred and transformed as it moves across actors and artifacts [34]. Each translation process has a *mode of transference* (how knowledge moves across actors/artifacts), and a *mode of transformation* (how knowledge changes as it is transferred). These different forms of translation processes mediate between and across actors and artifacts in several ways which we call: communicative-, automatic-, inscriptive- and retrieval translation. Below, we will discuss each translation process in detail (see Table 1 for a summary).

Table 1. Translation processes

Type of translation	Mode of transference	Mode of transformation
Communicative (Human-to-human)	Talking about code	Negotiation
Automatic (Artifact-to-artifact)	Running code	Computation
Inscriptive (Human-to-artifact)	Writing code	Materialization
Retrieval (Artifact-to-human)	Reading code	Cognition

3.1. Communicative Translation

Communicative translation refers to a process where social actors transfer knowledge representations using dialogical or conversational practices so that knowledge can be exchanged with other social actors (i.e. human actors). Quite simply, this is humans talking to other humans about code or software functions (mode of transference), so as to negotiate (mode of transformation) how the code should be configured, and what a particular solution means in particular contexts. In this process, the representational states are transformed from intangible knowledge forms residing in the minds of developers into expressions that make up conversations or discussions (which are often recorded and stored for future use), aimed to negotiate a shared vision of what the software is going to do, or how the code can be written, changed and improved. For instance, core developers usually have formal or informal meetings on the Internet before taking next big step. In these meetings, developers negotiate and mutually adjust to other's perspectives by means of conversation, discussion and debate so that cognitive frames of how to develop OSS will be aligned and disseminated across developer's boundaries. However, such a shared vision is not limited to discussions of the overall direction of software development, but can include a certain feature comment or even a tiny bug patch, depending on the communicative unit. Hence, communicative translation reveals the underlying mechanisms of social distribution whereby knowledge about developing OSS flows across actors.

3.2. Automatic Translation

Automatic translation is an automated and independent computational process that artifacts carry out to enable a knowledge flow from one representational state to another representational state. In such knowledge flows, formalized cognitive frames are transmitted through APIs and other standardized computational mechanisms through which different artifacts communicate with each other (mode of transference). As cognitive frames are transmitted, they are also transformed; their previous latent state is made explicit so as to reveal information that is relevant in different contexts (mode of transformation). This process assumes a particular way of structuring a set of artifact or services so as to enable transfer and transformation across artifacts. For example, in OSS development, testing code involves an automatic translation process. It transforms potential bugs within the OSS code into explicit warnings or error messages,

therefore, enabling representational states of errors and defects that are latently stored to be seen. The automatic translation process occurs autonomously without any intervention of human during the computation. Hence, automatic translation facilitates the dissemination of representational states across artifacts and reveals the underlying mechanisms of structural distribution whereby artifacts engaged in OSS development afford new workflows for knowledge distribution.

3.3. Inscriptive Translation

Inscriptive translation constitutes a process where individual actors materialize knowledge residing in human minds into external artifacts so that certain cognitive frames get inscribed into literal, graphical and code artifacts. Often this simply occurs through writing code (mode of transference), in such a way that socially held cognitive content is materialized (mode of transformation). Examples of such inscriptive translation may also include the ways in which bugs identified by developers are inscribed into a bug tracker, such as Github or Bugzilla, or how plans for an envisioned future are laid down in roadmaps on project websites. Inscriptive translation indicates that knowledge is transferred and transformed as it moves from social distribution to structural distribution. Hence, inscriptive translation distinguishes itself from communicative translation in that inscriptive translation emphasizes that cognition traverses the boundary between the social and the structural, and is materialized, while communicative translation focuses on formation of shared visions originating from interactions among social actors. Even though some communicative translation processes may lead to shared visions finally being situated in external artifacts, this last step cannot happen without the materialization that inscriptive translation affords.

3.4. Retrieval Translation

Retrieval translation refers to a process where implicit knowledge embedded in external artifacts is extracted and transformed by individual actors so as to ensure ensuing communicative, automatic or inscriptive translation processes. Essentially, humans read code and associated artifacts (mode of transference), in order to understand their significance, and hence retrieve relevant meanings stored in artifacts (mode of transformation). In OSS development, the retrieval activities can either relate directly to a certain artifact establishing basic constraints to which the project must conform (e.g. a C interpreter must conform to the C programming language), or may involve interplay of different artifacts acting as constraints [32]. Take forking for example; the forking project inherits the majority of requirements and source code from the forked OSS, which sets fundamental parameters for developing one's own, alternative, project. The developer has to understand existing functionalities and features of the forked project so as to modify the forked project with respect to his/her purpose. In so doing, retrieval translation transfers the implicit knowledge embedded in the forked project into the developer's mind. In addition, retrieval translation entails retrieval of the aforesaid archived knowledge inscribed in various artifacts for formulating new knowledge to develop OSS. Inverse to inscriptive translation, retrieval translation indicates knowledge flows from the structural distribution to the social distribution.

4. Four Modes of OSS Development

Translation processes form the micro-foundations of distinct modes of OSS development that emerge as particular configurations of translation processes, driven by the underlying social and structural distributions of a particular project. In this section we will show how the underlying social and structural distributions act as 'generative motors' that shape the emergence of discrete modes of OSS development processes, that characterize typical ways in which distributed cognitive processes are carried out in a certain project.

4.1. Distributions as Generative Motors

The particular ways in which OSS development processes unfold is deeply intertwined with the hierarchical form of the community and the set of artifacts they engage with. This mangle of social actors and material artifacts, each having distinct characteristics, produces an emergent stream of activities that we can capture as *typical modes* of OSS development. Such modes denote the ways in which distributed and abstract ideas are sourced, modified, integrated and made tangible over time in a distinct set of material artifacts, especially working code.

As *structural distributions* naturally vary across projects, we consider the different degrees of interpretative flexibility [2] of the set of material artifacts that make up the structural distribution. Interpretative flexibility describes the range of interpretations that are possible, when working with, and designing certain technological artifacts during the process of technological development [2]. The level of

interpretative flexibility determines the degree to which technological possibilities in OSS are many and diverse, or few and focused. Therefore, low levels of interpretative flexibility lead to projects being more constrained by features of specific technologies and standards which OSS must conform to, while high levels of interpretative flexibility give communities more freedom in how they combine and utilize heterogeneous artifacts while developing OSS projects. Granted, the set of standard artifacts that provide the base of a certain project can be replaced by a different set of standard artifacts that offers different technical possibilities. But as long as a specific set of artifacts is in place for a given project, it exercises a specific level of material constraint viz-a-viz the project, based on the level of interpretative flexibility of the structural distribution of which the artifacts are part.

Hence, the generative motor of a certain structural distribution is dependent on the interpretative flexibility of the artifacts that make up the said distribution. Lower levels of interpretative flexibility tend to lead to an iterative recreation of the structural distribution, whereas higher levels of interpretative flexibility open up possibilities for emergent combinations and development trajectories.

Further, OSS communities come together in different ways to translate distributed and abstract ideas into a specific set of software artifacts. Based on how knowledge and control are differently distributed, we can articulate how the *social distribution* of communities are configured so as to work together with a established structural distribution reflecting a certain level of interpretative flexibility. Hence, we propose that social distributions vary from oligarchy to democracy. Oligarchy denotes a situation where important knowledge is controlled and shared by several core developers, but where knowledge and control are highly concentrated and only part of them extend to the entire community. In contrast, democracy represents a situation where knowledge and control are distributed widely across those who contribute to the project. Therefore, the generative capacities of a social distribution can be distributed in different ways. A more centralized distribution will enable a few developers to more directly determine the direction of a project, whereas in more democratic distribution, project ambitions emerge as the result of dynamics between a large array of developers.

In Figure 1, an oligarchic social distribution is illustrated on the left, and a more democratic social distribution is illustrated on the right. From the figure, we can clearly discern how the oligarchic community clearly differentiates between core and periphery, whereas the democratic community has a larger set of

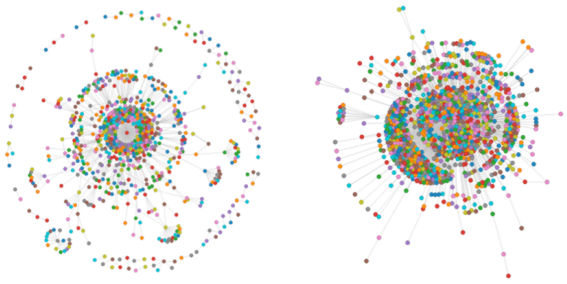

Figure 1. Social distributions

connections extending across all parts of the community.

Overall, a particular combination of social and structural distributions will shape the ways in which configurations of translations in OSS development processes unfold. Therefore, modes of OSS development capture the differing characteristics of such unfolding streams of distributed cognition. Below, we will discuss each mode of OSS development in greater detail (see Table 2).

Table 2. Four modes of OSS development

		Social Distribution	
		Oligarchy	Democracy
Structural Distribution	Low levels of interpretative flexibility	Type A: Technocracy	Type B: Mechanical Turks
	High levels of interpretative flexibility	Type C: Cathedral	Type D: Bazaar

4.2. Type A: Technocracy (Low levels of interpretative flexibility and Oligarchy)

Type A represents a mode of OSS development where knowledge flows and computation are centralized to a small number of developers and highly dependent on the features of specific artifacts. We choose to call this form of development *technocracy*, because it represents a situation where a small elite controls the software artifacts, not necessarily based on vision or ideas, but rather on fulfilling the technical promises of the underlying structural distribution. Twitter Bootstrap (see http://twitter.github.io/bootstrap/) is a typical such project, where a tight leadership core (two people) manages a product that largely capitalizes on stable Internet standards such as HTML, CSS, and Javascript. Hence, few people have decision-making power, and even their decision making power is heavily constrained by the aforesaid standards.

Due to the oligarchic social distribution, communicative translations seem to be less problematic in this form when compared to other

modes of OSS development. A consensus of OSS development tactics tends to be achieved easier here by a small number of developers. On the other hand, core developers are expected to be experts in the OSS related domain and standards are expected to sustain the viability of the project. Therefore, expert dominance and availability is a necessity for this type of project to succeed. Although the small core takes charge of the majority of the cognitive workloads associated with OSS development, the size of the periphery determines whether or not inscriptive translation processes will take place and how often. If the amount of peripheral developers is relatively large, such as Bootstrap, inscriptive translation processes are suggested by the periphery in the form of reported bugs and submitted patches, so that leadership visions can be easily inscribed into tangible artifacts (e.g., Bootstrap websites, roadmaps, issues) that help the periphery catches up with the leaders.

At the same time, due to low levels of interpretative flexibility, this mode of development aims to ensure compatibility with state-of-the-art standards and technology usage patterns. It may draw heavily on testing the OSS code on existing artifacts, and thus results in high frequencies of automatic translation. In addition, this mode requires large amounts of retrieval translation in order to exploit knowledge embedded in standards-based artifacts.

4.3. Type B: Mechanical Turks (Low levels of interpretative flexibility and Democracy)

Type B OSS development consists of a large number of developers who primarily conform to given parameters of present artifacts. We choose to label this form of development *Mechanical Turks* (inspired by Amazon's crowdsourcing service, and its namesake – an 18th century chess-playing automaton), due to the fact that in this mode a large group of people work in concert to build OSS based on a relatively fixed and constrained set of underlying material artifacts. VIM scripts (see http://www.vim.org/scripts/) are typical examples of this type of project. As the basic artifact, the fundamental function and form of the VIM text editor are largely fixed, a wide range of developers create different sets of VIM scripts geared towards enhancing the text and code editing functions of VIM. Most of them add new functionalities to the current version of the VIM text editor, even though their scope is limited. Examples include scripts for debugging code, syntax highlighting, and file managers accessible from within the editor itself. While the VIM editor does not allow for much flexibility in terms of what can be created, a large amount of smaller contributions reflecting the interests of diverse contributors in the community have created a wide array of scripts useful in various situations.

Because of the democratic distribution, communicative translations work to negotiate common images of the set of software artifacts that developers intend to deliver. But communicative translations can hardly dealt with the massive scale of knowledge flows if inscriptive translations are missing. As cognition is widely disseminated across a relatively large number of developers, the knowledge within developers' minds has to be inscribed into artifacts to enable effective and accurate sharing of information.

Low interpretative flexibility of the structural distribution reflects conformity in relation to standardized artifacts. It invites high levels of automatic translations, analogous to the technocracy mode described above, in order to maximize the compatibility between the project and basic artifacts upon which the project relies. As the role of technology and standardized artifacts becomes more pronounced, retrieval translations become increasingly important. The retrieval activities not only deal with certain standardized artifacts, but also invite retrieval of the archived knowledge inscribed in literal, graphical, and code forms.

4.4. Type C: Cathedral (High levels of interpretative flexibility and Oligarchy)

Type C OSS projects are developed by a small number of developers, but engage various heterogeneous artifacts. Project evolves from the visions of the leadership and may recombine artifacts in novel ways. We choose to label this form of development as a *cathedral*, based on Raymond's [24] classical fold to develop OSS. We argue that Linux kernel can be regarded as an example of a cathedral. Due to its clear management structure and self-referential material structure, the Linux kernel (see https://www.kernel.org/) shows how a leadership with a clear vision can shape a flexible underlying structural distribution into a valuable set of software artifacts. While many individuals contribute with ideas and suggestions to the Linux kernel, the actual decisions with regards to what code to implement is highly centralized to a small group, and eventually lies with the project founder himself. Since the Linux kernel relates directly to hardware, it is largely undetermined by other software artifacts. Therefore, there are many possible technical pathways that the project could proceed along, but the strict governance wielded by the project leadership corrals the project to stay on track within the boundaries of a particular pathway.

Communicative and inscriptive translation processes are at the forefront of the cathedral mode. Communicative translation processes help to ensure that a core leadership group agrees on fundamental principles of design and development, and facilitates to propagate such ideas throughout the community. However, communicative translation itself is insufficient for developers to trace the development trajectory given the high levels of uncertainty resulting from high levels of interpretative flexibility. Unless inscribing ongoing flows of knowledge in artifacts, project leadership can hardly control the overall development and anticipate various technological possibilities.

However, the possibility of either computing knowledge automatically via software artifacts or retrieving knowledge from basic artifacts is lower than in previous modes described above, in that a cathedral project lends the leadership more freedom in describing what the project should be, and what should or should not do during the development process. Therefore, such a process relies less on testing or on being compatible with the basic artifacts that the project draws upon (e.g. the CPU features that Linux bases its OS kernel upon provide a wide range of possibilities, and thus do not constrain the project excessively). Nevertheless, retrieval translations often occur in response to inscriptive translations by means of extracting archival knowledge that has been inscribed into artifacts.

4.5. Type D: Bazaar (High levels of interpretative flexibility and Democracy)

The most complex form of OSS is Type D where the social distribution is democratic and the structural distribution can be flexibly interpreted. We choose to call this form of OSS development a *bazaar,* because of its likeness with the ideal development process proposed by Raymond [24]. This mode of development is complex, and emerges from interactions of empowered developers who act on local concerns, with heterogeneous artifacts offering diverse technical opportunities. Rubinius (see http://rubini.us/), a Ruby Virtual Machine, represents a typical bazaar. While having a clear leadership team, control is effectively distributed through giving commit rights to anyone who has had a patch accepted. Further, the Ruby programming language is in constant flux, and multiple interpretations are possible, thus giving Rubinius a potential trendsetter role within the Ruby community.

Because of its emergent and unpredictable nature, a bazaar project requires a balanced and integrated configuration of communicative, automatic, inscriptive and retrieval translations. If such integration is achieved, the resulting development processes capitalize both on the needs and visions of a democratically distributed community, and the technical possibilities offered by structural distribution with a high level of interpretative flexibility.

5. Discussion

In response to the received notion of a monolithic and unitary mode of OSS development, we have proposed a taxonomic framework of OSS development modes and their micro-foundational translation processes. Such translation processes unfold differently, reflecting the underlying social and structural distributions that generate different cognitive activity patterns. Distinct combinations of different social and structural distributions, and the translation activities they execute, allow us to conceptualize four modes of OSS development. These modes constitute particular sociotechnical arrangements that translate requirements from distributed, uncertain and abstracted representations to functional and instantiated software artifacts. Below, we will discuss the implications of the proposed framework for the study and practice of OSS development, focusing on under what conditions a certain type of OSS project may succeed and how network- and sociotechnical scholars can make contributions to the understanding of a wide range of OSS development.

5.1. Governance Imperatives

The ways in which different projects are socially and structurally distributed and how such distributions shape the emergence of OSS development modes, create distinctly different conditions for ongoing success for each type of project (see Table 3 for a summary). We label these conditions *governance imperatives*, which are leadership, management, and organizational practices that need to be in place in order to increase the chances of successful computation of OSS requirements and therefore the overall success of the project. The logic of governance imperatives is that the ways in which activities are organized and supported, needs to match rather than contradict the ways in which social and structural resources are distributed. Below we will explain how each mode of OSS development has distinct governance imperatives.

Technocracy projects require a tight fit between the technological possibilities of the underlying structural distribution and the visions of its centralized leadership. If such a fit fails to realize, the project will try to achieve something very difficult, given the

underlying structural distribution. Therefore, it is likely to fail in its code development, or at least perform at suboptimal levels in relation to set goals.

Mechanical Turk projects are often fragmented and consist of individual developers who exchange development ideas based on their own, local concerns and perceived benefits, to arrange parts of the project in particular ways. However, such efforts are consistently constrained by the relevant artifacts at hand. In order to coordinate a large number of localized concerns, these projects require high levels of modularity in order to succeed. Such modularity ensures that a large number of developers can work simultaneously on multiple functionalities provided by the structural distribution, either because they are disconnected, or because their interfaces are conform to commonly available templates.

Cathedral projects are relatively unconstrained by their structural distribution, and therefore material constraints do not provide a clear direction. Therefore, cathedral projects require core developers to leverage exploration capabilities to search for better solutions among technical possibilities. Such dynamic activities are dependent on hierarchical governance and visionary leadership in order to maintain a certain direction despite diverse technological possibilities offered by structural distributions exhibiting high levels of interpretative flexibility.

Bazaar projects are fragile and prone to breakdowns (or rather, forking into several competing projects), since they have multiple possible development paths that are not necessarily restricted by either an underlying set of standards and artifacts or by a clear leadership vision. Therefore, they require expert project facilitation and consensus building processes so as to maintain cohesion and prevent splintering. Perhaps such projects are mostly valuable for exploration that lead to continuous splintering into smaller, more focused projects.

As we can see above, the first three modes (i.e., technocracy, mechanical Turks, and cathedrals) all have either low interpretative flexibility, or an oligarchic social distribution. Therefore their governance imperatives tend to derive mostly from the most constraining dimension (i.e. low interpretative flexibility or oligarchy), or through demanding a tight fit between possibilities offered by the structural distribution and the visions of the leadership (as is the case in technocratic projects). The most complex cases are the bazaar projects, which we argue require adept community leadership to manage the myriad technological possibilities offered by a flexible structural distribution, in concert with a widely empowered social distribution.

In summary, the governance imperatives described above suggest a wide range of conjectures that need to be explored further, both theoretically and empirically. Beginning to explore the various modes of OSS development, and the various conditions under which each can be successful, would represent a fruitful path forward for OSS research.

Table 3. OSS development modes and governance imperatives

Type	Description	Governance Imperative
Technocracy	Elitist core specifies a narrow technical vision	Fit between technological possibilities and leadership visions
Mechanical Turks	An evenly distributed community iterates on a technical base	Modularity and local benefits
Cathedral	Hierarchical organization working to stabilize sets of artifacts	Hierarchical governance and visionary leadership
Bazaar	Self-organized exploration of technical possibilities	Facilitating community consensus

5.2. Implications and Future Research

As we have argued in this paper, there are multiple modes of OSS development processes, and we urge scholars to stop seeing OSS as a unitary, monolithic mode of organizing software development. Rather, OSS projects can be characterized by underlying social and structural distributions. Such distributions shape emerging configurations of translation processes, which in turn shape the unfolding of distinct modes of OSS development. If we can recognize which mode of OSS development we are dealing with, we can more closely identify patterns in data and associations between characteristics of development processes and important outcome variables such as community growth and IS success [4]. Rather than searching for an elusive 'OSS mode of organizing', we can then more clearly discern the different factors that impact OSS processes and outcomes. Such inquiry into the different modes of OSS organizing can take the form of examining: 1) the underlying forces that generate different OSS processes, 2) how these processes differ, and 3) how they result in different outputs.

First, we have proposed ways in which social and structural distributions generate different modes of OSS development. While our argument is grounded in the literature, we also urge scholars to empirically examine the effects of these distributions. Further, we

also encourage scholars to uncover a larger array of other generative mechanisms that underlie OSS development processes and shape their unfolding.

Second, we have theorized that the ways in which OSS development processes unfold are varied. In doing so, we make a few first tentative steps towards providing a way of conceptually understanding this variation through proposing a typology. Going forward, scholars need to consider the ways in which OSS processes are conceptualized and contextualized, rather than assuming about a standard Raymondian process that all OSS projects follow.

Third, we have argued that there are good reasons to expect that the outcomes of different OSS processes will differ and that the conditions under which such outcomes are successful may vary. Therefore, OSS scholars need to think deeply about what the generative mechanisms inside of work processes are, that help to create desirable outcomes.

Noting the importance of the generative characteristics of not only social actors, but also artifacts, theorizing around concrete forms of material agencies is brought into focus. While scholars such as Leonardi [18] and Latour [17] have shed lights on the ways in which social and material agencies are *imbricated* or *hybridized*, the IS field still lacks a sophisticated theory of material agency on a par with those of social agency within sociology [7]. Careful theorizing of the generative capacities of material agencies would provide insight into how various structural distributions take part in the unfolding of IS development processes at large.

To enable scholars to understand how distinct modes of development differ in terms of antecedents, process characteristics, and associations with outcome variables, we specifically urge two camps of scholars to turn their attention to OSS phenomena: those who study networks and those who study sociotechnical systems. Combining the sociotechnical perspective with network methodologies would allow us to depict and analyze how particular combinations of structural and social distributions shape development processes differently, and create different associations with important covariates such as technical and social success [9].

We are keen to see both quantitative and qualitative studies of the evolution of sociotechnical networks in OSS, so as to understand how differently configured structural forms participate differently in unfolding OSS practices. Through using the large amounts of digital trace data available on OSS projects, interesting patterns could be uncovered. And deep ethnographic accounts may disclose the generative mechanisms behind these patterns, which then could be corroborated using quantitative network studies that show how relational practices evolve differently across various sociotechnical/material networks.

6. Conclusion

As scholars increasingly recognize the considerable variation present in OSS modes of organizing, it is critical to examine how such different modes emerge. Our paper takes initial steps in this direction through formulating micro-foundations consisting of translation processes and revealing configurations of underlying social and structural distributions. Future empirical work needs to expand the proposed theoretical model with careful analysis of the distributive and processual dynamics of different development processes, and how different forms of participation in these processes are subject to different conditions of success.

7. References

[1] R.R.P. Bagozzi, and U.M. Dholakia, "Open Source Software User Communities: A Study of Participation in Linux User Groups". Management Science, 52(7), 2006, pp. 1099–1115.

[2] W.E.Bijker, T.P. Hughes, and T.J.Pinch, The social construction of technological systems: New directions in the sociology and history of technology, MIT Press, 1987.

[3] A. Capiluppi, A. Faria, and J. Ramil, "Exploring the Relationship between Cumulative Change and Complexity in an Open Source System", Proceedings of the Ninth European Conference on Software Maintenance and Reengineering (CSMR'05), 2005.

[4] K. Crowston, J. Howison, and H. Annabi, "Information systems success in free and open source software development: theory and measures", Software Process: Improvement and Practice, 11(2), 2006, pp. 123–148.

[5] K. Crowston, and J. Howison, "The social structure of free and open source software development", First Monday 2, 2005.

[6] 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, 49(6), 2007, pp. 564–575.

[7] M. Emirbayer, and A. Mische, "What Is Agency?", American Journal of Sociology, 103(4), 1998, pp. 962–1023.

[8] M. Godfrey, "Evolution in open source software: a case study", In Proceedings International Conference on Software Maintenance (ICSM-94), 2000, pp. 131–142.

[9] R.Grewal, G.L. Lilien, and G. Mallapragada, "Location, Location, Location: How Network Embeddedness Affects

Project Success in Open Source Systems", Management Science, 52(7), 2006, pp. 1043–1056.

[10] E. von Hippel, and G. von Krogh, "Open Source Software and the “Private-Collective” Innovation Model: Issues for Organization Science", Organization Science, 14(2), 2003, pp. 209–223.

[11] E. Hutchins, and T. Klausen, "Distributed Cognition in an Airline Cockpit", In Y. Engeström, and D. Middleton, eds., Cognition and Communication at Work, Cambridge University Press, 1996, pp. 15–34.

[12] E.Hutchins, Cognition in the Wild, MIT Press, Cambridge, MA, 1995.

[13] E. Hutchins, "How a Cockpit Remembers Its Speeds", Cognitive Science, 288, 1995, pp. 265–288.

[14] E. Hutchins, "Distributed Cognition. In International Encyclopedia of the Social and Behavior Sciences", Elsevier, 2001, pp. 2068–2072.

[15] S. Koch, "Evolution of open source software systems–a large-scale investigation", Proceedings of the First International Conference on Open Source Systems, Genova, 11th-15th July, 2005, pp. 148–153.

[16] S. Krishnamurthy, "Cave or Community?: An Empirical Examination of 100 Mature Open Source Projects", First Monday, 2, 2002.

[17] B.Latour, We Have Never Been Modern, Harvard University Press, Cambridge, Massachusetts, 2012.

[18] P. Leonardi, "When Flexible Routines meet Flexible Technologies: Affordance, Constraint, and the Imbrication of Human and Material Agencies", MIS Quarterly, 35(1), 2010, pp. 147–167.

[19] A. Mockus, R.T. Fielding, and J.D. Herbsleb, "Two case studies of open source software development: Apache and Mozilla", ACM Transactions on Software Engineering and Methodology, 11(3), 2002, pp. 309–346.

[20] N. Bezroukov, "Open Source Software Development as a Special Type of Academic Research (Critique of Vulgar Raymondism)", First Monday, 4(10), 1999.

[21] J. Noll, and W.-M. Liu, "Requirements Elicitation in Open Source Software Development", Proceedings of the 3rd International Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development, ACM Press, 2010, pp. 35–40.

[22] M. Perry, "The Application of Individually and Socially Distributed Cognition in Workplace Studies: Two Peas in a Pod?", Proceedings of European Conference on Cognitive Science, 1999, pp. 87–92.

[23] E. Raymond, "The cathedral and the bazaar", Knowledge, Technology & Policy, 12(3), 1999, pp. 23–49.

[24] E.Raymond, The cathedral and the bazaar: musings on Linux and Open Source by an accidental revolutionary, O’Reilly, 2001.

[25] G. Robles, J.J. Amor, J.M. Gonzalez-Barahona, I. Herraiz, U. Rey, and J. Carlos, "Evolution and Growth in Large Libre Software Projects", Proceedings of the 2005 Eighth International Workshop on Principles of Software Evolution (IWPSE’05), 2005, pp. 1–10.

[26] Y. Rogers, and J. Ellis, "Distributed Cognition: An Alternative Framework for Analysing and Explaining Collaborative Working", Journal of Information Technology, 9(2), 1994, pp. 119–128.

[27] W. Scacchi, "Understanding the requirements for developing open source software systems", IEE Proceedings Software, 149(1), 2002, pp. 24-39.

[28] W. Scacchi, "Socio-Technical Interaction Networks in Free/Open Source Software Development Processes", In S. Acuña, and N. Juristo, eds., Software Process Modeling, Springer, 2005, pp. 1–27.

[29] W. Scacchi, "Understanding Requirements for Open Source Software." In Lyytinen, K., Loucopoulos, K., Mylopoulos, J., and Robinson, B., eds., Design Requirements Engineering: A Ten-Year Perspective, Springer, Berlin, Germany, 2009, pp. 467–494.

[30] S.K. Shah, "Motivation, Governance, and the Viability of Hybrid Forms in Open Source Software Development", Management Science, 52(7), 2006, pp. 1000–1014.

[31] S. Valverde, and R. Solé. "Self-organization versus hierarchy in open-source social networks", Physical Review E, 76(4), 2007, pp. 1–8.

[32] D. Wegner, "Transactive memory: A contemporary analysis of the group mind" In Mullen, B., Goethals, G. R., eds., Theories of Group Behavior, Springer, New York, 1987.

[33] Y. Ye, and K. Kishida, "Toward an Understanding of the Motivation of Open Source Software Developers", Proceedings of the 25th International Conference on Software Engineering", 2003, pp. 419–429.

[34] Y. Yoo, K. Lyytinen, and R.J. Boland, "Distributed Innovation in Classes of Networks", Proceedings of the 41st Annual Hawaii International Conference on System Sciences, 2008.