Modularity and Organizational Dynamics in Open Source Software (OSS) Production

Eunyoung Moon **James Howison**
School of Information
University of Texas at Austin
eymoon@utexas.edu, jhowison@ischool.utexas.edu

Abstract

Modularity has been seen as key to the success of OSS projects. However empirical studies on modularity of OSS systems have resulted in confusing results. To account for underlying mechanisms of those confusing results, we systematically examine widely studied OSS projects. Based on our systematic review on technical and organizational structures, we suggest that organizational circumstances of OSS production are at least in a continuum of tightly-coupled and loosely-coupled organizational circumstances of production in which both geographically distributed volunteers and paid developers with organizational ties work together (albeit separately over time). Furthermore, organizational circumstances of OSS production appear to be dynamic, as firms move in and out of OSS production communities over time. In essence we argue that the reason for the confusing empirical results was a persistent assumption that organizational circumstances of OSS production are static or unitary; rather what matters is the organizational circumstances of production in any episode of contribution. This research agenda paper proposes future inquiries to develop a comprehensive picture of ecological shift in different levels of system modularity and organizational circumstances of OSS production over time and through episodes.

Keywords

Modularity, organizational dynamics, open source software, production communities

Introduction

The relationship between the structure of organizations and the structure of products they produce has been of key interest, both in Information Systems and beyond. Investigating this relationship has led to valuable, actionable, insights both in traditional organizations and in studies of product development and innovation outside corporate boundaries (e.g., Baldwin and Clark, 2000; Parnas 1972b; MacCormack et al., 2006). In particular, the relationship between software structure and the organization of software development has been seen to be linked to the success of open source development (Benkler and Nissenbaum, 2006; Crowston et al., 2012; Lerner and Tirole, 2002; O'Reilly 1999; Torvalds 1999). As traditional business organizations increasingly interact with open development, any insight into how to successfully work with open projects is needed (e.g., Fitzerald 2006; Germonprez et al., 2013).

Modularity has been seen key to the success of OSS projects. Both researchers and practitioners have argued that OSS should be modularized to enable distributed developers to work on the project independently without having to affect other's tasks. Or, to put it another way, theory and practice has argued that successful OSS projects must develop in a modularized manner since open development does not provide the ability to overcome the substantial coordination requirements of developing in a non-modularized manner. Without modular design approach, the argument runs, open collaboration would break down, resulting in failures. Therefore, successful projects must have developed in a modular fashion.

However, empirical studies on modularity of OSS systems have provided confusing results. For instance, the concrete architecture of the Linux kernel is almost fully connected, which implies that modules of the

Linux kernel are tightly coupled to each other (Bowman et al., 1999). The Linux kernel is found to have "unsafe dependencies" between kernel modules and non-kernel modules, and researchers raised potential concerns about the maintainability of the Linux kernel (Yu et al., 2004). On the other hand, others found that Linux has relatively less interdependence between modules, compared to commercial software product, the first version of Mozilla (MacCormack et al., 2006). Yet, in that same paper, MacCormack et al (2006) found that Mozilla was successful in modularizing their codebase later, and argue that this led to an increase in volunteer participation, supporting the idea that modularity is key to open development.

Colfer and Baldwin (2010) confirm the confusing results about open development and modular production structure. They reviewed 102 studies of the relationship between organizational structure and technical structure, drawn from the organizational science literature, both in open projects and in closed software development. Overall, they found strong support for what they refer to "the mirroring hypothesis": that organizational and product structures match each other (and ought to for successful collaboration). This high overall support, however, was a result of very strong support in studies within firms and studies across firms. Yet, in the area of open collaboration they found low support for the hypothesis, only finding support in 5 of the 16 studies they examined. In the other 11 studies, they found that the organizational structure did not match with the product structure, yet development has been successful.

In this paper, we review empirical results linking product and organizational structures in open source and agree that they are confusing, leaving this promising approach somewhat stuck. Beyond the contribution of Colfer and Baldwin (2010), we provide a different explanation for why the results appear confused and propose a relatively straightforward conceptual solution and empirical approach that ought to advance research in this area.

Literature review

We review studies of three important open source projects, examining findings on modularity and asking about respective organizational forms by considering the extent of corporate participation in OSS production community. We chose to examine studies of the Linux kernel, Apache, and GNOME, each large and successful open source project which has been frequently studied (Crowston et al., 2012). We note at the outset that each of these production communities have significant corporate participation; they fall into the category of Open Source 2.0, as outlined by Fitzgerald (2006).

Having selected particular OSS projects to examine, we searched for studies investigating those projects from the conferences and journals in management and information systems, software engineering, and organizational studies. This paper aims to contrast and combine outcomes across disciplines, hoping to identify patterns of what we are seeing with respect to modularity and propose theoretical explanations for them. The following section provides details of relevant studies in a succinct way due to the limitation of the space.

Linux kernel

The development of the Linux operating system was begun by Finnish computer science student Linus Torvalds in 1991. The Linux project has attracted several thousands of developers and other contributors from all over the world (Hertel et al., 2003)

Hybrid mode in Linux kernel development

The survey of 141 Linux kernel developers revealed that both paid developers and volunteers co-exist in the Linux kernel project (Hertel et al., 2003). In that early survey, it was reported that 20 percent of the developers were paid for their Linux programming work on a regular basis and another 23 percent at least sometimes, whereas the remaining 57 percent of participants received no salary at all for their work on the Linux kernel.

More recent analysis of the Linux kernel releases reported that since 2005 nearly 10,000 individuals from more than 1,000 distinct companies have contributed to the Linux kernel (Corbet et al., 2013). It also reported that a large portion of Linux kernel work is performed by developers using their corresponding

companies' email address. To be specific, this analysis reported that 1,392 kernel developers and 243 identifiable companies have contributed to the version of Linux kernel 3.10. Moreover, it reported that over half of the patches for the kernel pass through a handful developers from four companies, based on the analysis of kernel subsystem maintainers' affiliation. It also reported that more than 80% of all kernel development is performed by developers who are being paid for their work, whereas unpaid developers account for 13.6 percent of kernel developers.

Embedded Linux is one example that is mainly developed by commercial firms with a relatively small number of hobbyists (Henkel 2006). According to Henkel (2006), embedded Linux is the area in which nearly all developments come from commercial firms and few volunteers truly engage. Embedded Linux plays a significant role in embedded software and is used in embedded devices including mobile phones and machine controls. Commercial firms in embedded software engage in the development of modules or extensions to operate Linux for embedded systems. The survey of individuals working on those modules reveals that nearly all developments come from firms such as device manufacturers, component manufacturers, and dedicated software firms, whereas volunteers account for small portion.

To be specific, the analyses of 96 kernel versions reveal corporate participation within the major subsystems such as *arch* (Godfrey and Tu, 2000). In that study, it was found that a large amount of stable code was contributed by external third parties. For instance, the source code for IBM S/390 mainframe (Vepstas) was integrated into the stable release. More specifically, figure 1 shows a large amount of lines of codes (LOCs) suddenly increases in the *arch* subsystem. In figure 1, each line represents respective subsystem within *arch* subsystem. Since *arch* subsystem contains all platform-specific code including i386 and s390, there is each subsystem per a specific architecture within *arch* subsystem (Mosberger and Eranian, 2001). For example, in figure 1 *arch/i386* represents the lines of codes that are specific to the Intel 386 architecture. Although the very first version of Linux was targeted at the Intel 386 architecture, later that code specific to the Intel 386 architecture was moved from the main *kernel* subsystem into *arch/i386* in order to simultaneously support a specific tree for any other different machine architecture (Antoniol et al., 2002). The first sudden upward leaps occurred due to this movement of the code in early 1995. However, figure 1 shows that "the sudden upward leaps" were made in many subsystems within *arch*. More importantly, those cases of "one large lump" in LOCs were found to have been developed and maintained by independent teams of developers and external developers from corporations (Godfrey and Tu, 2000). However, unfortunately, few studies have attempted to look at the impact of large amount of code from external corporate parties on modularity of the Linux kernel at the system level as well as within the subsystem level.

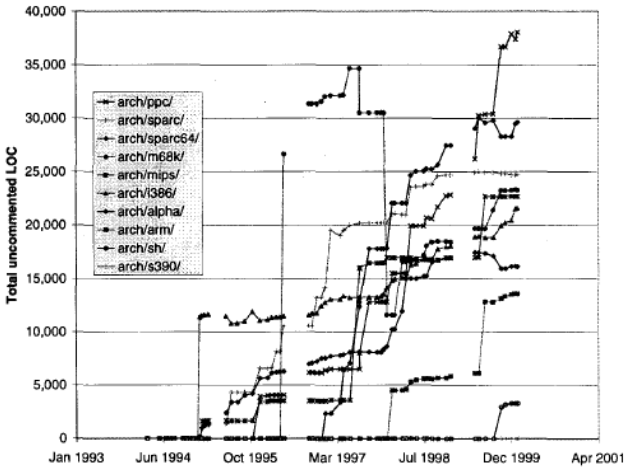

Figure 1. The Growth of the *arch* subsystem of the Linux kernel

(Source: Figure 10 from Godfrey and Tu, 2000)

Taken as a whole, there has been a continual corporate involvement in Linux kernel development. Paid developers from commercial firms have increasingly accounted for the significant portion in producing Linux kernel, whereas there has been a decline in the portion of volunteer participation. This indicates

that organizational dynamics have occurred in Linux production; the organizational form in Linux production has dynamically changed over time. In short, the Linux project is considered a classic example of corporate participation in open source production (Germonprez et al., 2013).

System design of Linux kernel

The Linux kernel operating system consists of a large number of modules that are arranged around the kernel (Hertel et al., 2003). The Linux kernel itself is a large system and responsible for process, memory, and hardware device management (Bowman et al., 1999). While the Linux kernel is the essential part of the Linux operating system, individual applications, libraries, resources and auxiliaries function as the extensions of the kernel's functionality (Lokhman et al., 2013) (see Figure 2).

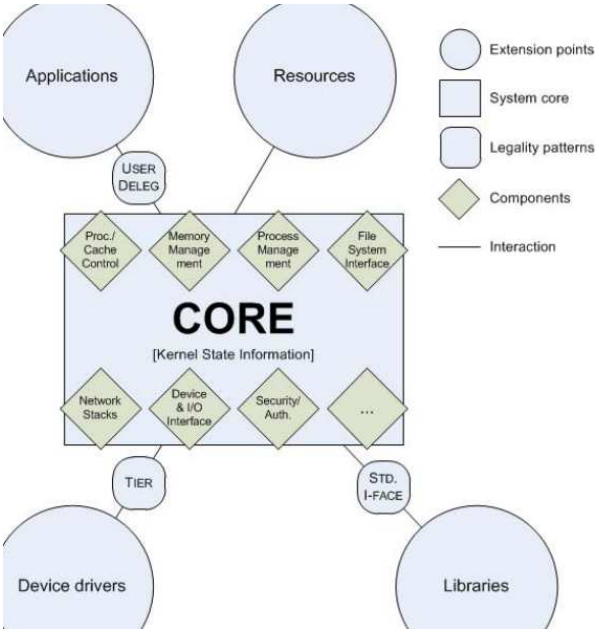

Figure 2. The architecture of the Linux kernel (Source: Figure 2 from Lokhman et al., 2013)

The conceptual architecture and the concrete architecture of the Linux kernel revealed substantial discrepancy (Bowman et al., 1999). While the conceptual architecture shows "how developers think about a system", the concrete architecture shows "the relationships that exist in the implemented system" (Bowman et al., 1999). That is to say, the concrete architecture can also be called "as-built architecture". The conceptual architecture of the Linux kernel was determined, based on the analyses of existing documentation that describes an architectural description of the structure. This analysis revealed that the Linux system was implemented, following implementation-hiding principle. In contrast, the concrete architecture, which was extracted by a source-code extractor (Gall et al., 1995), revealed that it is, in fact, almost completely connected at the highest level of abstraction. This analysis indicates that modules of the Linux kernel are tightly coupled to each other.

The investigation on common coupling of the Linux kernel counted the number of instances between kernel modules that are common to all installations and non-kernel modules that are included in specific installations (Yu et al., 2004). That analysis identified that most kernel modules reference at least one global variable and only three kernel modules have no common coupling with any other kernel modules. Based on categorization of common coupling, the authors in that study pointed out that Linux contains a number of "unsafe dependencies" between kernel modules and non-kernel modules through the use of global variables, suggesting that those dependencies need to be eliminated for the maintainability of the Linux operating system in the long term.

A comparison between the Linux operating system and the Mozilla web browser (MacCormack et al., 2006) found that Linux had a relatively more modular architecture than the predecessor of Mozilla. However, Netscape released the code under an open source license and renamed it as Mozilla later, as a

part of strategy for increasing competition in the browser market. Mozilla purposefully underwent a redesign for the purpose of capturing contributions from volunteer developers. As a consequence, Mozilla become significantly more modular than its predecessor as well as Linux and furthermore it led to increase volunteer participation.

In summary, empirical investigation on system design of the Linux kernel reveals that modules of the Linux kernel have been densely connected to each other, although degree of modularity varies according to the specific version and specific period. Moreover, our review reveals that external corporations significantly started to engage in Linux development from late 90's by contributing to a large amount of stable code, while there has been a smooth decline of volunteer participation over time. However, unfortunately, we know relatively little about whether and how firm's dynamic involvement in kernel development relates to different degree of modularity.

Apache project

The Apache web server and associated projects are under the umbrella of the Apache Software Foundation (ASF) and considered successful open source products (Hann et al., 2002). By the Apache projects in this paper, we mean all the projects under the ASF.

Hybrid mode in the Apache projects

The Apache group is described as "businesslike development teams" in the world of OSS with the advent of the World Wide Web and was full of people in the business of building the web (Wayner 2000). Several members of core developers formed their own companies and used the code for their products. Also, big corporations such as IBM had begun to engage in Apache implementations and hired one of the key developers. Furthermore, IBM engineers contributed code back to the Apache project (West 2003). The study on paid developers' participation in the Apache project also reported the existence of paid developers and there is an increased attention from corporations who pay their employees working on the Apache projects (Roberts et al., 2006).

To be specific, the study on the Apache web server project reported that more than 100 "feature-filled modules" were distributed by third parties and many of those modules contain more lines of code than the core server in the Apache server project (Mockus et al., 2002). The Apache-SSL project is taken as an example of numerous those satellite projects, started by interested third parties to add unique functionality targeting a particular group of users.

In brief, the Apache projects have ad significant corporate involvement and firm's involvement in the Apache projects appear to respond to the environmental changes such as the advent and the popularity of the World Wide Web.

System design of the Apache projects

The relationships among modules in the Apache projects and dual case of those among developers were investigated (Lopez-Fernandez et al., 2006). For the analyses of the relationships among modules, modules network was devised and defined "two modules are linked together by an edge when there is at least one committer who has contributed to both" (Lopez-Fernandez et al., 2006). Those edges were weighted, using a degree relationship, defined as the total number of commits performed by common committers between two modules. The analysis of modules network revealed that Apache modules are only linked to 8% of other modules in the versioning system. Based on this analysis, it was suggested that Apache developers likely reach some points in which they do not have to get to know each other since the Apache project is fragmented into many families of modules.

The analysis of the Apache web server project identified that it comprises core modules and peripheral modules (Lokhman et al., 2013). This is also congruent with the analysis of the Apache web server projects that revealed it has a core/periphery structure (MacCormack et al., 2010). Furthermore, it was found that the Apache web server project comprises a small core part that must be present in every installation and a larger body of peripheral code that is used for varying features by extending core functionality (MacCormack et al., 2010).

To summarize empirical studies on modularity of the Apache projects, the Apache projects appear to have high degree of modularity, although degree of modularity varies according to the specific versions and periods. When we take a closer look at the Apache server project specifically, it also shows that the project was particularly designed with modular approach. In addition, the Apache server project shows it comprises a small group of core modules and a larger group of peripheral modules. While the Apache projects have had significant corporate participation, few studies attempted to further investigate whether and to what extent corporate involvement in the Apache projects is mirrored in the modularity of the Apache projects.

GNOME

The GNU Network Object Model Environment (GNOME) project aims to build a GUI desktop environment and application framework for Unix systems (de Icaza 1999). GNOME comprises of numerous modules and this paper examines its specific modules, *Evolution* and *Gnumeric*. *Evolution*, the default email client for GNOME, was started in 1999 and has been widely used in OSS communities (German 2004). *Gnumeric*, a spreadsheet application, was initially started by de Icaza, the founder of the GNOME project.

Hybrid mode in GNOME development

The GNOME project was founded by Miguel de Icaza in 1996 with a loosely coupled group of developers, distributed all over the world. GNOME has attracted the attention of software companies and, firms have been actively engaging in GNOME development. Simultaneously, volunteer have also created enterprises, expecting to sell services and products around GNOME (German 2002). For instance, a commercial venture, Helix code (Ximian later) was started by de Icaza, aiming to continue the development of GNOME and planning to generate income by selling services around GNOME (German 2004). Ximian had taken over the development of *Evolution* and committed several employees worked on the development of *Evolution*. Developers of *Evolution*, *Gnumeric*, and many other smaller applications were employed by Helix. On the other hand, paid developers from Eazel (now bankrupt) were responsible for file manager *Nautilus* module. In addition, the GNOME project has had significant corporate investment from firms that distribute software as a component of the Linux operating system and other firms that employ GNOME as a base toolkit for designing and manufacturing embedded devices such as PDAs and mobile phones (Wagstrom et al., 2010). The most 10 active developers of GNOME appeared to be Ximian employees or consultants (German 2003). Paid developers from Red Hat concentrated on the development of several critical modules of GNOME such as *gtk+*, *ORBIT*, and *CORBA* (German 2003). Most of work in GNOME has been concentrated among a small number of developers paid to contributed to the project by private firms. Those developers had responsibility for such tasks as project design and coordination, testing, documentation, and bug fixing (German 2002; 2003) to make sure the continuation of GNOME production.

Taken as a whole, the GNOME project shows dual worlds of volunteers and paid developers in OSS production (Wagstrom et al., 2010). Although GNOME, originally comprised only of a handful of volunteers, has had significant corporate involvement. In addition, the core modules of GNOME were found that they were had been mainly concentrated by paid developers from commercial firms.

System design of GNOME

GNOME comprises a large collection of programs and libraries (German 2002). Specifically, the official distribution of version 2.4 contained 76 different modules including applications such as a spreadsheet (*Gnumeric*), a mail client (*Evolution*), and a word processor (*Abiword*) (German 2003).

Evolution had been consciously modularized from the beginning of the project and it had the seven largest modules (German 2004). *Gnumeric* is found that it has a significantly higher propagation cost, compared to other open source products—*Gnucash, Abiword*, Linux, and MySQL—when investigated by MacCormack et al (2012). That is to say, the degree to which a change to any single element of *Gnumeric* causes a potential change to other elements in the system appears higher than the counterpart of other open source products. The analysis of *Gnumeric* application revealed that it has a larger density of

dependencies than even commercial software product with similar size and function, and furthermore, its largest module had most of these dependences.

While *Gnumeric* application is found to break the mirroring hypothesis, this anomaly seems to be resolved by further review on what types of organizational forms, in fact, have engaged in *Gnumeric* production. As reviewed earlier, *Gnumeric* development had been mostly concentrated by a handful of key individuals employed by private firms (German 2002). MacCormack et al (2012) also confirm that *Gnumeric* application is not developed by loosely-coupled organizational forms, based on the analyses of credits file and *Gnumeric*'s change log.

Modularity in a hybrid mode of OSS production

As the success of OSS has attracted firms interested in making profits from OSS production, corporately employed developers have increasingly engaged in OSS production communities. A recent study on the ratio of paid developers and volunteers in OSS production (Riehle et al., 2014) reported that approximately 50 percent of all OSS production has been paid work. Prior studies also provided findings that volunteers and paid developers from numerous firms work together (Godfrey et al., 2000; Hertel et al., 2003; Roberts et al., 2006; West 2003) and furthermore, some particular modules of OSS systems have been mainly developed and maintained by paid developers sponsored by commercial firms (German 2003; Henkel 2006; Wagstrom et al., 2010). For instance, as the Linux community matured, corporations such as Red Hat started engaging in the community and have provided Linux-related products and services that were not supplied by the community (Lerner and Tirole, 2002). Corporate participation in OSS production communities has expanded rapidly over the past 15 years (Germonprez et al., 2013).

This emergent phenomenon demonstrates how OSS work is increasingly shifted into a commercially feasible form, which is labeled as "OSS 2.0" by Fitzgerald (2006). This emergence of external involvement in OSS production shows that OSS production communities increasingly comprise both volunteers and paid developers simultaneously work on OSS development (Langlois and Garzarelli, 2008).

Despite firm's dynamic involvement in OSS production communities, prior studies on modularity of OSS systems tend to have an assumption that organizational circumstances of OSS production are both static and unitary. This assumption, which was primarily held by prior studies on modularity of OSS systems, resulted in confusing findings on modularity. Accordingly, it's not clear whether and to what extent corporate participation in OSS production communities significantly affects different degree of OSS modularity. While particular modules of OSS systems are found to have been mainly developed and maintained by corporately employed developers, prior studies on modularity of OSS systems have mainly looked at modularity of OSS systems with assumptions that the organizational circumstances of OSS production are static and remain loosely-coupled at one extreme, rather than that they both can change over time and might be different for different contributions as the involvement of firms and volunteers winds in and out of the project over time.

A look towards the future

The initial evidence that was obtained as a result of our review of empirical studies has conceptual implications, which are highlighted in this section.

The proprietary and open source software development models are polarized at the extremes of strategies, along a variety of dimensions. For instance, open source software projects are characterized by a number of geographically dispersed volunteers working on the project independently (Raymond 2001). This mode of organizational form is considered loosely-coupled (Orton and Weick 1990; Weick 1976). On the other hand, organizational form to develop proprietary software is at another extreme. Pre-assigned sets of individuals are often co-located at one site and dedicated towards explicit and shared goals. Accordingly, it has been assumed that the development of OSS systems cannot be as tightly planned and managed as most commercial software, then modularity in the codebase becomes even more important for the success of OSS systems. Each module of OSS systems can be produced in dependently without having to affect other's tasks and thus, OSS work can be incremental and asynchronous (Benkler 2006). In other words, OSS products are claimed to be more modular than commercial software products (MacCormack et al., 2012; O'Reilly 1999; Raymond 2001), mirroring its organizational circumstances of production. Following

this line of reasoning, researchers have long argued that these contrasting organizational forms are reflected in the architecture of software that they produce (e.g., Colfer and Baldwin, 2010; MacCormack et al., 2012).

While those claims indeed have provided useful insights into two contrasting organizational forms in software development, our review on three large-scale OSS projects reveals that organizational forms of OSS systems are at least mixed forms of two extremes. While our review mainly focused on three large-scale OSS projects widely studied, Colfer and Baldwin (2010)'s test of mirroring hypothesis that organizational ties will be mirrored in technical dependencies in OSS projects also exhibits that many cases clearly challenged the hypothesis. In that study, it was found that many open collaborative groups (11 studies out of 16 studies) do not show modular systems. If we look at this results with a 'static' view of organizational forms in OSS production, it may seem to be inexplicable. As we illuminated in our analyses of empirical studies, however, those prior studies primarily assumed that OSS products developed by geographically distributed individuals with few organizational ties are unlikely tightly planned. Accordingly, it has been argued that OSS products must be modular to attract contribution from a number of distributed individuals independently working on OSS production since modular design enables those individuals to contribute to OSS production without having to learn the whole systems or affecting other parts of systems.

However, our review reveals that some parts of OSS systems have been mainly developed and maintained by groups of paid developers from commercial firms. Commercial firm's involvement in OSS production appear to be fluid, responding to external environments in which the products are used. Embedded Linux, the subsystem *arch* of the Linux kernel, Apache-SSL, *Evolution*, and *Gnumeric* are typical examples of OSS production in a hybrid mode. The subsystem *arch* had a significant amount of stable code from external corporate (Godfrey et al., 2000). *Gnumeric*, a module of GNOME, was also identified as one case that was developed by corporately employed developers (German 2002; MacCormack et al., 2012), resulting in significantly higher propagation cost other than other open source products and a larger density of dependencies than the commercial software product.

Since memberships are fluid in OSS production communities, both volunteers and paid developers can move in and out of the community and thus, the organizational circumstances of production are unlikely to be at one extreme. The fluidity of open development belies efforts to relate organizational structure and product structure because, unlike formal organizations, there may be no particular structure that persists over time. Rather the product results from episodes of participation (Annabi et al, 2008) and each episode may involve a production environment that is quite distinct from the production environment of other episodes; the 'organization' is not unitary, thus the organizational circumstances of production are not unitary. In this sense, it makes little sense to ask about overall relationships; inevitably this leads to a confusing 'averaging' of real production environments that obscures rather than reveals how the work was done. On the other hand, we argue, it makes a great deal of sense to ask about the relationship between episodes of participation, the way those episodes of work were organized, and the product structure before, during, and after that participation.

Given that modularity of OSS systems appears to vary in specific periods in which corporate involvement dynamically occurs, we need additional analysis to investigate whether and to what extent modules of OSS systems were purposefully modularized and concentrated by corporate investment. In light of this, we propose the following future research question, which asks, "How does tightly-coupled participation impact the modularity of OSS contributions and systems?" In investigating this question, we may first consider specific periods when corporate involvement occurred and which modules were particularly developed and maintained by corporately employed participants (especially when a corporate participant might be an interface between the open project and a tightly-coupled internal team). Based on this, we may track whether and to what extent modularity of those specific modules have varied, as firms come in and out of OSS production communities.

Following this relatively simple re-casting of inquiry between organizational and product structure, we believe that researchers can ask more interesting questions. For example, we propose the following questions, "If corporate involvement lowers modularity of OSS systems (thus potentially creating problems for ongoing open development), how do OSS communities respond? Or we may identify episodes of corporate participation that appears to fit well with the open source way of working, raising the very interesting question of how has the firm (or its employees) generated and sustained that

approach to community development? That is to say, paid developers from commercial firms are tightly-coupled since they are dedicated to software development towards explicitly shared goals. However, their participation in OSS communities may not be necessarily tightly-coupled, even if the organizations themselves are tightly coupled internally. For instance, Wagstrom et al (2010) found two distinct types of firms' strategies, community-focused approach and product-focused approach in GNOME community. Community-focused firms package the entire output of a community, for instance Linux distributors, while product-focused firms typically have interests in particular components from a community, for example, a component library or a particular application in relation to their business model. By breaking down participation into episodes, rather than looking for atemporal relationships between organization and product structure, we could see how these different strategies relate to open development and the resulting structure of open products.

Conclusion

This paper examined studies of technical and organizational structures in frequently studied large-scale OSS projects. The work examined in this paper was pulled from diverse disciplines including management and information systems, software engineering, and organizational studies for the purpose of identifying and synthesizing patterns of what each discipline has been seeing with respect to modularity. Each discipline has provided both quantitative and qualitative data. However, as Colfer and Baldwin (2010) also found, the empirical results are somewhat confusing; the findings tend to be fragmented and show conflicting results rather than being closely weaved into a synthesized picture of the role of modularity in OSS production.

We hope to move this debate forward with a simple observation: the production of code in open systems occurs over time and implies many different circumstances of production. In particular, it is insufficient to characterize open collaborations as having a particular organizational structure: products produced through open collaboration are the aggregate of work undertaken over time (Howison and Crowston, 2014); and these episodes occur in many different organizational circumstances. And it is the circumstances of production, not some atemporal overarching organizational structure, that was always held to be related to product structure. What is needed, we argue, is a way of thinking about organizational dynamics of OSS production communities—particularly as corporate participation winds in and out—and whether and how system modularity responds to those dynamics. This relatively simple shift in approach ought to lead to much more interesting, and practically, relevant, results.

REFERENCES

Annabi, H., Crowston, K., and Heckman, R. 2008. "Depicting What Really Matters: Using Episodes to Study Latent Phenomenon," in *Proceedings of the International Conference on Information Systems (ICIS)*.

Antoniol, G., Villano, U., Merlo, E., & Di Penta, M. (2002). Analyzing cloning evolution in the linux kernel. *Information and Software Technology*, *44*(13), 755-765.

Baldwin, C. Y., & Clark, K. B. (2000). *Design rules: The power of modularity* (Vol. 1). Mit Press.

Benkler, Y., & Nissenbaum, H. (2006). Commons-based Peer Production and Virtue*. *Journal of Political Philosophy*, *14*(4), 394-419.

Bowman, I. T., Holt, R. C., & Brewster, N. V. (1999, May). Linux as a case study: Its extracted software architecture. In *Proceedings of the 21st international conference on Software engineering* (pp. 555-563). ACM.

Colfer, L., & Baldwin, C. (2010). The mirroring hypothesis: Theory, evidence and exceptions. *Harvard Business School Finance Working Paper*, (10-058).

Conway, M.E. (1968). How do committees invent. Datamation, vol.14, no.5, pp.28-31.

Corbet, J. , Kroah-Hartman, G., & McPherson, A. (2013). Linux Kernel Development – How fast it is going, who is doing it, what they are doing, and who is sponsoring it? (2013 Edition).

Crowston, K., Wei, K., Howison, J., & Wiggins, A. (2012). Free/Libre open-source software development: What we know and what we do not know. *ACM Computing Surveys (CSUR)*, *44*(2), 7.

de Icaza, M., Lee, E., Mena, F., & Tromey, T. (1998, April). The GNOME desktop environment. In *Proceedings of the annual conference on USENIX Annual Technical Conference* (pp. 38-38). USENIX Association.

Fitzgerald, B. (2006). The transformation of open source software. *Mis Quarterly*, 587-598.
German, D. M. (2002, May). The evolution of the GNOME Project. In *Proceedings of the 2nd Workshop on Open Source Software Engineering* (pp. 20-24).
German, D. M. (2003). The GNOME project: a case study of open source, global software development. *Software Process: Improvement and Practice*, *8*(4), 201-215.
German, D. M. (2004). Using software trails to reconstruct the evolution of software. *Journal of Software Maintenance and Evolution: Research and Practice*, *16*(6), 367-384.
Germonprez, M., Allen, J. P., Warner, B., Hill, J., & McClements, G. (2013). Open source communities of competitors. *Interactions*, *20*(6), 54–59. doi:10.1145/2527191
Godfrey, M. W., & Tu, Q. (2000). Evolution in open source software: A case study. In *Software Maintenance, 2000. Proceedings. International Conference on* (pp. 131-142). IEEE.
Hann, I. H., Roberts, J. A., Slaughter, S., & Fielding, R. T. (2002, December). Economic Incentives for Participating in Open Source Software Projects. In *ICIS* (p. 33).
Henkel, J. (2006). Selective revealing in open innovation processes: The case of embedded Linux. *Research policy*, *35*(7), 953-969.
Hertel, G., Niedner, S., & Herrmann, S. (2003). Motivation of software developers in Open Source projects: an Internet-based survey of contributors to the Linux kernel. *Research policy*, *32*(7), 1159-1177.
Howison, J., and Crowston, K. 2014. "Collaboration through open superposition: A theory of the open source way," *MIS Quarterly* (38:1), pp. 29–50.
Langlois, R. N., & Garzarelli, G. (2008). Of Hackers and Hairdressers: Modularity and the Organizational Economics of Open-source Collaboration. *Industry and Innovation*, *15*(2), 125-143.
Lerner, J., & Tirole, J. (2002). Some simple economics of open source. *The journal of industrial economics*, *50*(2), 197-234.
Lokhman, A., Mikkonen, T., Hammouda, I., Kazman, R., & Chen, H. M. (2013, January). A Core-Periphery-Legality Architectural Style for Open Source System Development. In *System Sciences (HICSS), 2013 46th Hawaii International Conference on* (pp. 3148-3157). IEEE.
López-Fernández, L., Robles, G., González-Barahona, J. M., & Herraiz, I. (2006). Applying social network analysis techniques to community-driven libre software projects. *International Journal of Information Technology and Web Engineering (IJITWE)*, *1*(3), 27-48.
MacCormack, A., Rusnak, J., & Baldwin, C.Y. (2006). Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code. Management Science, 52, 7, pp.1015-1030.
MacCormack, A. (2010, August). The architecture of complex systems: do" core-periphery" structures dominate?. In *Academy of Management Proceedings* (Vol. 2010, No. 1, pp. 1-6). Academy of Management.
MacCormack, A., Baldwin, C., & Rusnak, J. (2012). Exploring the duality between product and organizational architectures: A test of the "mirroring" hypothesis. *Research Policy*, *41*(8), 1309-1324.
Mockus, A., Fielding, R. T., & Herbsleb, J. D. (2002). Two case studies of open source software development: Apache and Mozilla. ACM Transactions on Software Engineering and Methodology (TOSEM), 11(3), 309-346.
Moon, J. Y., & Sproull, L. (2000). Essence of distributed work: The case of the Linux kernel. *First Monday, 5*(11).
Mosberger, D., & Eranian, S. (2001). *IA-64 Linux kernel: design and implementation*. Prentice Hall PTR.
O'Reilly, T. (1999). Lessons from open source software development. *Communications of the ACM*, *42*(4), 33.
Orton, J. D., & Weick, K. E. (1990). Loosely coupled systems: A reconceptualization. *Academy of management review*, *15*(2), 203-223.
Parnas, D.L. (1972a). Information distribution aspects of design methodology. *Information Processing* 71: 339-344.
Parnas, D. L. (1972b). On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, *15*(12), 1053-1058.
Raymond, E. S. (2001). The Cathedral & the Bazaar: Musings on linux and open source by an accidental revolutionary. O'Reilly.
Riehle, D., Riemer, P., Kolassa, C., & Schmidt, M. (2014). Paid vs. Volunteer Work in Open Source. In *Proc. of the 47th Hawaii Int'l Conf. on System Science (HICSS)*.

Roberts, J. A., Hann, I. H., & Slaughter, S. A. (2006). Understanding the motivations, participation, and performance of open source software developers: A longitudinal study of the Apache projects. *Management science*, *52*(7), 984-999.

Schilling, M. A. (2000). Toward a general modular systems theory and its application to interfirm product modularity. *Academy of management review*, *25*(2), 312-334.

Torvalds, L. (1999). The linux edge. *Communications of the ACM*, *42*(4), 38-39.

Vepstas, L. Linux on the IBM ESA/390 mainframe architecture. Website. http://linas.org/linux/i370/i370.html

Yu, L., Schach, S. R., Chen, K., & Offutt, J. (2004). Categorization of common coupling and its application to the maintainability of the Linux kernel. *Software Engineering, IEEE Transactions on*, *30*(10), 694-706.

Wagstrom, P., Herbsleb, J. D., Kraut, R. E., & Mockus, A. (2010, August). The impact of commercial organizations on volunteer participation in an online community. In *Academy of Management Annual Meeting*.

Wayner, P. (2000). Free for all: How Linux and the free software movement undercut the high-tech titans. New York: Harper Business.

Weick, K. E. (1976). Educational organizations as loosely coupled systems. *Administrative science quarterly*, 1-19.

West, J. (2003). How open is open enough?: Melding proprietary and open source platform strategies. *Research Policy*, *32*(7), 1259-1285.