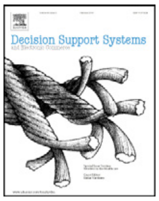
Decision Support Systems 80 (2015) 27–41

Contents lists available at ScienceDirect

Decision Support Systems

journal homepage: www.elsevier.com/locate/dss



Taxing the development structure of open source communities: An information processing view

Mohammad AlMarzouq [a,1], Varun Grover [b,*,2], Jason Bennett Thatcher [b,3]

[a] Department of Quantitative Methods and Information Systems, College of Business Administration, Kuwait University, PO Box 5486 Safat 13115, Kuwait
[b] Department of Management, College of Business and Behavioral Sciences, Clemson University, Clemson, SC 29634, United States

ARTICLE INFO

Article history:
Received 25 June 2014
Received in revised form 25 July 2015
Accepted 14 September 2015
Available online 9 October 2015

Keywords:
Free/Libre and Open Source Software
FLOSS
Organizational Information Processing Theory
Software development
Modularity
Brooks' law

ABSTRACT

Committers in Free/Libre and Open Source Software (FLOSS) projects shoulder responsibility for evaluating contributions and coordinating the broader community development effort. Given committers' central role in development processes, we examine whether how they are organized influences FLOSS community performance. Specifically, drawing on the lens of Organizational Information Processing Theory (OIPT), we develop a model that explains how committal a structure's ability to manage information impacts FLOSS community performance. Based on archival data drawn from 237 active FLOSS communities, we found that the performance of centralized and decentralized FLOSS communities varied with three conditions tied to information flows: task routineness, uncertainty and task interdependence. Our empirical results support the idea that FLOSS communities performing development tasks that are generally routine, highly interdependent, and generate little contributor uncertainty will perform better under a centralized committal structure. On the other hand, decentralized committal structures thrive under the conditions of task non-routineness, low task interdependence, and high contributor uncertainty. We conclude with a discussion of results, limitations, and directions for future research.

1. Introduction

Free/Libre and Open Source Software (FLOSS) communities are self-organizing, online groups of developers who create freely available software products. Within FLOSS communities, while contributors write source code patches that implement a feature or fix a bug, committers review their submissions and approve their integration into the community source code base [81]. Committers are responsible for directing individual members' development efforts as well as coordinate their output's integration into a software package [74]. Often, committers are promoted from the ranks of contributors who have demonstrated superior levels of competence as well as dedication to a project. Even though contributors and committers are not directly compensated, FLOSS communities often create applications that are equivalent to, or supplant, commercially available software. These products are increasingly used by individuals and organizations to complete essential tasks [31,86].

That a large, self-managed group of distributed volunteer developers can develop such high quality software [4] defies the conventional wisdom of software engineering. Brooks' Law [12] suggests that as the number of developers grows, ramp up effects make conventional software projects more inefficient and ineffective due to delays tied to communication and coordination costs. For example, when an additional developer is added, Brooks argues that production is slowed, and errors more likely, while that developer acquires knowledge about the codebase. One might question, based on Brooks' Law, the viability of large FLOSS projects where the size and number of participations might hinder progress. However, large FLOSS projects like the Linux Kernel are thriving in ways that suggest we still have more to learn about the software development process [48].

To explain FLOSS communities' efficacy, open source developers argue that unique software development practices, explain their ability to coordinate activities as effectively as for-profit developers. Raymond [81] argues that because the source code conveys rich information, FLOSS developers must communicate less to develop projects. Moreover, Raymond argues that the FLOSS philosophy of releasing software updates early and often, provides opportunities for early correction of bugs and more incremental changes in the software, thereby reducing developers' need to directly communicate frequently. Raymond described FLOSS communities as a bazaar for ideas, and FLOSS advocates contend that these bazaars can compete with traditional software development companies.

However, not all FLOSS communities are created equal. Krishnamurthy [47] found that the majority of FLOSS communities were highly centralized. Moreover, researchers have found substantial

* Corresponding author.
E-mail addresses: almarzouq@mis.cba.edu.kw (M. AlMarzouq), vgrover@clemson.edu (V. Grover), jthatch@clemson.edu (J.B. Thatcher).
[1] Tel.: +965 24988630.
[2] Tel.: +1 864 656 3773.
[3] Tel.: +1 864 656 3751.

http://dx.doi.org/10.1016/j.dss.2015.09.004
0167-9236/© 2015 Elsevier B.V. All rights reserved.

variation in FLOSS communities' structure and efficacy [20,23,64]. Interestingly, FLOSS research offers empirical support that some FLOSS communities match Raymond's description of a bazaar [44,85] while others are more centrally controlled and observe similar traits as traditional software teams described by Brooks [13]. The fact that both views are represented in practice, suggests that much remains to be learned about how to optimally structure FLOSS communities such that they develop better software products.

To better understand FLOSS community's performance, this study delineates contingency factors regarding developmental tasks (routineness, uncertainty, inter-dependence) and hypothesizes how they can influence the ability to successfully develop code. We then examine how the structure of FLOSS communities changes this relationship. Specifically, we examine how the emergent committal structures within FLOSS communities relate to their ability to create and update software. In order to frame our study, we leverage insights from Organizational Information Processing Theory (OIPT) [33] to help reconcile Brooks and Raymond's views on performance. The remainder of this study unfolds as follows. First, we review FLOSS community structure and identify different committal structures. Then, we introduce OIPT as a potential explanation for variance in FLOSS community performance. Drawing on OIPT, we develop a model that suggests FLOSS community performance reflects the fit between the development task and the development structure. Next, we discuss the methods used to collect data and evaluate our research model. Finally, we discuss the results, implications, and limitations of our study and offer directions for future research.

2. Free/Libre and Open Source Software (Floss) Communities

FLOSS communities have been described as knowledge-sharing and production communities [54]. To integrate members' voluntary contributions into software, FLOSS communities rely on emergent leaders and coordination processes [73]. Leaders self-select, or are picked by existing leaders from the membership, based on their abilities and interests in tasks that they perform [10,24,54,81,89]. Moreover, because most FLOSS members participate for a short time [74,89], FLOSS communities' coordinating structures tend to change over time [72].

Although dynamic, FLOSS community structure can be inferred from patterns of member participation. In this study since we infer structural attributes from the activity of members, it is useful to contextualize this work with a brief description of the nature of the community's structure and membership. Crowston and Howison [23] describe FLOSS community structure as onion shaped with four layers of members: the core, the periphery, active users, and passive users (see Fig. 1). Core members are formal members of the original development team and often perform more frequent and consistent development work than periphery members [23,88]. Active and passive users are merely consumers of the FLOSS community product; however, active users do contribute feature requests and bug reports to the developer community [104].

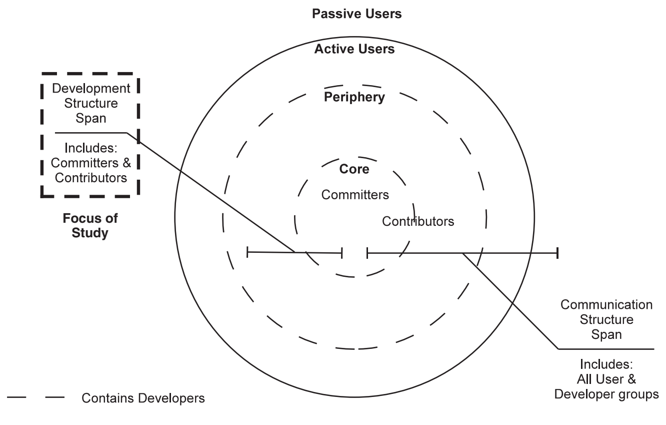

Passive Users
Active Users
Periphery
Core
Committers
Contributors
Development Structure Span
Includes: Committers & Contributors
Focus of Study
Communication Structure Span
Includes: All User & Developer groups
Contains Developers

Fig. 1. FLOSS community structure (adapted from [23]).

FLOSS communities are bound by how development and communication structures shape member's interaction. The development structure organizes the activities of core and peripheral developers. The communication structure spans every layer of a FLOSS community to convey information on the software [23,64]. Within the development structure (core and peripheral), members play two roles: committer and contributor. Both types of members contribute to software development. However, the committers also possess access rights to the community code base. As a result, committers can incorporate changes directly into the community code base, while contributors have to work with a committer to do so. Committers rise from the ranks of the contributors after they have proven their trustworthiness and technical competence through their continued contributions [82,89].

Committers are the busiest FLOSS community members. Not only are they themselves developers, they are also tasked with making decisions about other contributors' work [89]. For example, if a contribution is accepted, the committer is tasked with integrating the contributed patch into the code base, which places more responsibility and work on the committer's shoulders, especially when the committed code breaks the work of other developers.

Because of committers' responsibility for assessing and integrating all contributions, they represent potential bottlenecks in the FLOSS development process (cf. [37]). To illustrate that this is a problem that many FLOSS communities face, we present excerpts from the guidelines of some well known FLOSS projects in Table 1. These community guidelines (in Subversion, Mozilla, or Apache), suggest that committers' capacity to manage contributions drives the FLOSS development process and by understanding how committers are structured to manage their workload, we may glean insight into FLOSS communities' performance [37].

Committal structure refers to how workload is distributed among the committers. It is a result of a conscious community decision related to who is given authority to commit code changes rather than a result of deciding to use FLOSS development tools [61], and is assumed to reflect the centralization tendencies in the community [36]. FLOSS communities are centralized when committers are a small group relative to the development structure's membership. In an extreme case, a community with only one authorized committer would be highly centralized. The committal structure is decentralized when committers represent a larger

Table 1
Evidence of delays in FLOSS the development process.

Community	Excerpt	Notes
Subversion	If you don't get a response for a while, and don't see the patch applied, it may just mean that people are really busy.	Patch committals can experience delays.
Mozilla	Getting attention: If a reviewer doesn't respond within a week or so of the review request: • Join #developers …	Because delays in the review process are all too common, the Mozilla community has a process for how to deal with the problem.
Apache	What if my patch gets ignored? Because Apache has only a small number of volunteer developers, and these developers are often very busy, it is possible that your patch will not receive any immediate feedback. Developers must prioritize their time, dealing first with serious bugs and with parts of the code in which they have interest and knowledge. Here are some suggestions on what you can do to encourage action on your patch:…	Delays in patch committal are all too common and the community explains the reasons and gives suggestions on how to alleviate the problem.

portion of the developers. Thus, the opposite extreme case would be a community where every user could commit a patch to the code base would be considered decentralized.

Although Raymond [81] describes FLOSS communities as decentralized babbling bazaars, FLOSS communities display centralized (e.g. [23,47]) and decentralized structures [23,64]. However, the implications of these structures for FLOSS community performance are not fully understood. While descriptive work exists, relatively little theory driven work has examined FLOSS committal structure and productivity [88]. To understand committal structure's implications for FLOSS performance, we draw on Organizational Information Processing Theory (OIPT) (e.g. [33,97,98]). OIPT has the potential to yield insight into the circumstances under which different committal structures will lead to FLOSS community performance. OIPT is particularly relevant to understanding committal structures' implications because it directs attention to organizational structures and information flows as predictors of organizational performance.

3. Floss and Organizational Information Processing Theory

Organizational Information Processing Theory (OIPT) [33] suggests that fit between information-processing capabilities and information-processing demands shapes FLOSS communities' performance. It is particularly relevant to FLOSS communities because it offers an explanation for how committal structures influence the knowledge acquisition and exchange necessary to effectively coordinate [30,54] changes in a projects' code base [21]. Therefore the structure influences the speed of software development. OIPT logic suggests that a FLOSS community's decision to be centralized or decentralized will be paramount to performance. Furthermore, OIPT suggests that there are other contingency factors that are unique to each FLOSS community, which should be considered when choosing the optimal structure for the community.

FLOSS communities' effectiveness rests on their ability to facilitate internal information transfers [33]. For example, when contributors lack knowledge required to make a contribution, such as identifying which files need to be changed in order to implement a feature or resolve a bug, they will seek out that knowledge. The committers, having experience with the source code and the community in general, will be the members most likely to possess this knowledge or could at least identify individuals who do. Therefore, from an OIPT perspective, FLOSS community effectiveness hinges on contributors and committers ability to transfer information necessary to resolve the myriad of issues tied to software development.

To structure internal information transfers, OIPT implies that FLOSS communities will develop hierarchies. Although the relationship between committers and contributors is not one between a superior and a subordinate in a conventional software organization (e.g., manager and employee), completing a development task in FLOSS communities requires them to interact in a hierarchical manner. Before a contribution is committed into the community code base, committers are required to review and, possibly, modify patches before they are committed. As one committer puts it: “I've written before on mailing lists that only about two out of every five submitted patches I review go in unchanged on a good day and that seems to match other maintainers' experiences, too,[4]” indicating that a hierarchy guides the development of FLOSS communities.

According to OIPT, FLOSS communities suffer when their hierarchies' structure no longer “fit” the information processing needs of their members. For example, when the number of requests exceeds the committers' capacity for handling them, contributions are not processed as quickly, committers are overloaded, and the overall performance of the community suffers. This uncertainty is inherent in an organization populated by volunteers working at their own pace. It reflects a lack of fit between the FLOSS community's information processing needs and its information-processing capacity [33,97].

It is likely that such lack of fit could occur in FLOSS communities, since committers are mostly volunteers and will very likely have less time to dedicate to community coordination than would a full-time supervisor in a more conventional organization [29]. Furthermore, whatever time committers can afford to give to the community will not be entirely dedicated to resolving contributor issues, as the committers will also be engaged in both their own development tasks and interactions with the broader community [18,24,64,89].

To ensure fit, OIPT suggests that FLOSS communities, like organizations, can choose from two different fit strategies to optimize information flows and improve development performance. First, a FLOSS community could increase the information-processing capacity of the committal structure, by adding more committers, thereby giving them more time to process contributions as well as free up time for their own development activities and improving the overall development performance of the community. Second, the community could take action to reduce their information processing needs by limiting the number of committers to experts who can ensure compatibility across patches. To be effective, OIPT suggests that fit strategies must mitigate uncertainty tied to information transfer in FLOSS communities [33].

4. Research model

Leveraging insight from OIPT, we turn to developing a model of FLOSS community information processing needs, structure, and performance. Our thesis is simple. We argue that contingency factors determine the information processing needs of the FLOSS community that shape performance. These contingency factors[5] are: task routineness, defined as the unpredictability of the task, contributor uncertainty, defined as the number of contributions from peripheral developers, and task interdependence, defined as the extent to which tasks require coordination to be accomplished. Moreover, we investigate whether the centralization of the development structure moderates ties from these sources of information processing needs to performance. Below, we first describe the outcome construct (performance) followed by the antecedent constructs and the OIPT based logic behind our hypotheses in more detail. The constructs are described in Table 2 and the research model is illustrated in Fig. 2.

4.1. Performance of FLOSS communities

Inherently, FLOSS communities can be viewed as information processing communities whose goal is successful software development. When considering software development projects and performance, it is important to distinguish between effort and progress [12]. Even when software developers expend substantial effort on a project, it may not achieve sufficient quality to be released to market. For example, in later stages of a project, Brooks [12] argues that the effort needed to bring new developers up to speed (i.e., the ramp-up effect) and coordinate development slows the rate at which more productive developers can add new functionality to an application, thereby introducing inefficiencies and slowing its progress toward market. Hence, from Brooks' view, progress (i.e., meeting deadlines to create or meeting requirements) is an essential metric of a software development project's performance [21,27,28,35,70].

In some respects, the dynamics that influence FLOSS community performance are similar to traditional software development teams. Not unlike ramp-up effects described by Brooks [12], FLOSS community committers incur learning costs when they assist new contributors, by answering their technical questions and giving them guidance [101]. Also, not unlike lead developers in traditional development teams

[4] From the blog of a committer in the Django web framework FLOSS project: http://www.pointy-stick.com/blog/2007/11/02/development-experiences-version-control/.

[5] These are classic contingency factors extensively used in an IP context. They represent a task's information needs based on the work of Perrow, and Thompson, among others.

Table 2
Overview of theoretical constructs.

Construct	Definition
Performance	Progress made toward meeting the demands of the FLOSS community.
Centralization	The degree to which the committal activity in a FLOSS community is concentrated in the hands of a small group of committers relative to the overall size of the development structure's membership
Task routineness (simplicity)	The degree of predictability in the software-development task done within the FLOSS community.
Contributor uncertainty	The unpredictability in development tasks that is introduced from the need to integrate the work from contributors who are considered external to the FLOSS development structure.
Task interdependence	The degree to which developers' development tasks in a FLOSS community require cooperation.

[12], committers incur coordination costs with each patch committed to the community codebase. These costs vary depending on the number of contributors or other committers with whom the focal committer must interact as well as the amount of code affected by a proposed patch. When learning and coordination costs are high, committers have less time to develop new patches and contribute their own patches to the community codebase.

Despite these similarities with traditional software development teams, it is difficult to evaluate FLOSS community performance using conventional software development metrics. Traditional teams are evaluated in terms of their ability to meet deadlines and fulfill project requirements [12]. Because participation is voluntary, FLOSS communities are rarely able to set and enforce deadlines [81]. Also, where traditional teams work to fulfill preset development goals on a timeline, FLOSS communities frequently redefine goals, with no guarantee of when or if desirable features will be implemented [84]. Given these constraints tied to voluntary, self-directed members, conventional software development project performance measures are problematic at best, and inappropriate at worst, for evaluating FLOSS community performance.

Rather than conventional performance measures, FLOSS communities' performance must be estimated based on metrics tied to community productivity. A reasonable measure of FLOSS productivity is the total commits of patches to a community code base [45]. A commit signals that a patch has been successfully incorporated and contributes to progress toward meeting a community goal [40] and only occurs after the committer has processed relevant information and is satisfied with the quality of the contribution [101]. From an OIPT perspective, a greater number of commits suggest that a FLOSS community effectively manages learning and coordination costs associated with information transfers tied to contributions and commits. That is, size of project and number of committers being equal, a high performing FLOSS community will commit more patches to its codebase.

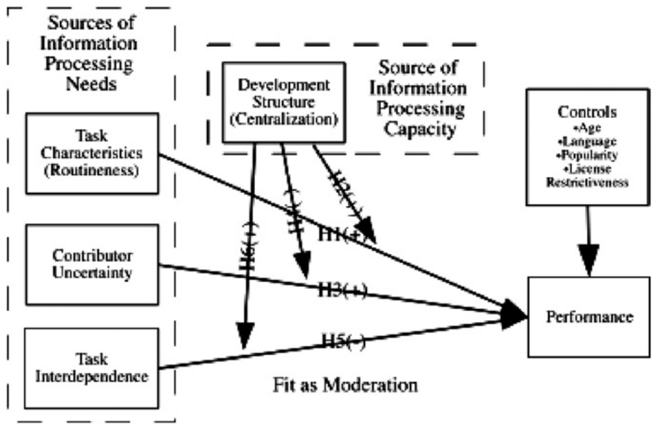

Sources of Information Processing Needs
Task Characteristics (Routineness)
Contributor Uncertainty
Task Interdependence
Development Structure (Centralization)
Source of Information Processing Capacity
Controls •Age •Language •Popularity •License Restrictiveness
H1(+)
H2(+)
H3(+)
H4(+)
H5(-)
H6(+)
Performance
Fit as Moderation

Fig. 2. Research model.

4.2. Development structure in FLOSS communities

How FLOSS communities process information and coordinate tasks influences their ability to generate commits and create software [21,74]. The committal structure shapes how these activities are performed as well as the when commits occur. The committal structure's information processing capacity shapes how much effort committers must expend (i.e., coordination and learning) to commit patches. To understand FLOSS community performance, it is necessary to examine how its committal structure relates to projects' progress.

We consider how centralization of a FLOSS community's committal structure influences the number of commits. In the organization theory literature, centralization has been conceptualized as a supervisor's span of control [41]. Centralized structures place responsibility for communicating with, and overseeing, a large group of employees in the hands of a small group of supervisors. In centralized structures, supervisors are prone to being overwhelmed by the demands of communicating with, and managing, their employees [9]. In decentralized structures, supervisors communicate with, and oversee, a smaller group of employees. Due to their narrower span of control, supervisors in decentralized structures are able to dedicate more of their time to each employee [9].

Even though the relationship between committers and contributors might not be that of supervisor and subordinate, centralization effectively describes communication between the two within a FLOSS community. A patch may require that a committer communicate with a large number of contributors and coordinate their activities, which results in a situation that is analogous to having a wide span of control (i.e., centralized structure). When communication demands are too broad, they may overwhelm the committers' capacity for information processing as the number of contributors increase. A narrower span of control (i.e., decentralized structure) on the other hand is less likely to overwhelm any individual committer and spread the workload over a greater number of committers.

Within the FLOSS context, centralization reflects the proportion of community members who possess committal rights. Centralization can be measured as the ratio of committers relative to the overall size of membership. Decentralized structures are expected to provide a greater capacity for processing information, since a larger proportion of committers will not only add to the processing capacity of the structure but will distribute more of the workload across the community. However, we do not expect that adding committers will be costless (cf. [98]) since a larger group of committers could increase the need for coordination [12,21]. As a result, consistent with OIPT, we believe it is necessary to examine contingencies which increase information processing needs and affect the committal structure's ability to handle information transfer in FLOSS communities.

4.3. Contingency factors that increase IP Needs: Task routineness (simplicity)

Task routineness (simplicity) refers to predictability in FLOSS community processes [60,79,97]. We use the term task routineness, but it also could be described as task simplicity. Routine tasks are simple in nature [96]. Simple tasks require that developers possess a limited domain of knowledge and exert little effort to complete [38,105]. In contrast, non-routine tasks have a great degree of unpredictability [79] and require complex knowledge to complete [96]. As a result, developers put forth greater effort to complete non-routine tasks [38,105].

Patches may create routine or non-routine work for committers. A routine or simple patch is one that requires few changes in the community codebase. Since routine patches require little knowledge to complete, they require less communication between contributors and committers when compared to non-routine patches. Therefore, we expect routine patches to put little strain on information processing capability of the committal structure. In contrast, non-routine patches require mastery of greater knowledge of the codebase. A non-routine patch is one that makes extensive changes in the community codebase in order to solve a problem or add functionality. To effectively develop such a patch, a contributor must understand the codebase's structure and the implications of adding patches for other files. Such knowledge is necessary, because good programming practice require one to encapsulate similar functionality into the same modules or files [76]. Given that FLOSS communities generally try to adhere to good programming practices [58,81], non-routine patches require committers to possess knowledge on the breadth of a FLOSS project.

Beyond requiring knowledge of the codebase, routine and non-routine patches create coordination costs for a committal structure. As the number of files modified by a patch increase, a committer must also review a correspondingly greater amount of code. In addition, as more files are impacted, the likelihood that such a complex patch will tie to the work of other developers' increases, thereby requiring committers to coordinate the work of more developers. Furthermore, since the code is not localized to a specific file, when a patch introduces a bug, the committer's debugging task will be much more complex since he/she is required to trace a larger number of files. As a result, we expect that non-routine patches will place greater coordination demands on committers, thereby placing a heaver information-processing burden on the committal structure than would routine tasks.

Furthermore, non-routine patches place higher learning costs on a committal structure. Contributors who create non-routine patches will change a greater portion of the source file in the code base. Before a change is made to any file, the contributor must understand the contents of that file and understand the best way to make the changes. To obtain that understanding, contributors will not only need to read the source code [81], but they will also need to seek the assistance of committers in helping them understand the impact of their changes [49]. As a result, greater information-processing requirements and learning costs are placed on the committal structure due to learning requirements associated with committing non-routine patches. Hence, we propose:

H1. Task routineness is positively related to a FLOSS community's performance.

Centralized committal structures may be well suited for processing routine patches [53]. A centralized committal structure, despite its limited information processing capacity, will have fewer coordination costs tied to committing patches. In a centralized structure, a committer will be more aware of the changes being made throughout the codebase as well as their implications for overall FLOSS software. As a result, committers will need to communicate with few contributors or other committers, if at all, when considering whether to commit a patch to the codebase. In contrast, decentralized structures will introduce extra overhead due to the need to coordinate with other committers (cf. [98]). This is because committers will need to communicate with each other as well as contributors about the implications of even relatively modest change in the code base. Therefore, we expect FLOSS communities with centralized committal structures to perform at higher levels as development tasks are more routine.

When patches are non-routine, decentralized committal structures may result in more productivity. Learning and communication costs associated with non-routine patches may increase in centralized development structures [100]. In terms of learning, non-routine patches require committers to have more specific knowledge of one part of the code-base or tacit knowledge of how a patch will relate to the overall codebase. In terms of communication, non-routine patches require committers to communicate with contributors about complex patches; but also, invest time in communicating changes with other committers. In this context, a distributed committal structure, which allows committers to specialize in different aspects of complex code base development [38], may be more effective. With a greater number of specialized committers, information-processing demands are distributed across the community, resulting in less effort being performed by any one individual [97]. Coordination and learning activities can then be performed in parallel, resulting in less overall time needed to perform them and more patches being committed to the community codebase. Hence,

H2. As tasks become more routine, FLOSS communities with centralized committal structures will have higher performance gains than communities with decentralized committal structures.

4.4. Contingency factors that increase IP needs: Contributor uncertainty

Contributor uncertainty refers to unpredictability in development processes introduced by adding new FLOSS community members. New contributors introduce uncertainty in two ways. First, new contributors introduce variability in learning costs. Before new contributors can submit patches, they need to acquire knowledge about the code base. While some of the knowledge is explicit, e.g., one can read the code, contributors often seek advice on how the project is structured and issues that need to be addressed from more experienced community members [101]. Because the committers are the most knowledgeable about the code base, they invest time in communicating with contributors, responding to their questions, and explaining the intricacies of the code base [101], which translates to higher information processing demands on community's committal structure. Second, similar to Brooks' [12] ramp-up effects, new contributors introduce uncertainty about the quality of patches. This is particularly true of contributors who are new to the FLOSS development process, as more experienced contributors are likely have absorbed tacit norms about, and knowledge necessary to make, contributions to the community. In addition, new contributors are likely to be the least knowledgeable about the code base. Due to new contributors' lack of knowledge about norms and the code base, their patches are more likely to introduce bugs, violate some of the programming guidelines, or fail to adhere to assumptions, such as variable access rules and coding conventions. Therefore, patches from new contributors will require more scrutiny and, therefore, more effort. Therefore, we expect contributor uncertainty to have a negative impact on performance. Hence,

H3. Contributor uncertainty will be negatively related to a FLOSS community's performance.

Centralization may influence contributor uncertainty's influence on FLOSS community performance. In centralized communities, committers may become a bottleneck for processing commits. When committers take time to teach new contributors the codebase's details, they are drawn away from developing code as well as committing patches. Moreover, when a problem occurs during a patch's committal, a committer has to read through the code base to find the bug or communicate with other developers to find a solution. As such demands on committers' time increases, a centralized committal structure will become overwhelmed due to its limited information-processing capacity [2,97]. In contrast, decentralized communities have more committers available to share knowledge needed by new contributors. Also, because decentralized communities have a higher proportion of committers, they are better able to handle demands of debugging flawed patches. Where a centralized community could come to a grinding halt while key committers debug a flaw, committers in decentralized communities may work in parallel on different patches or bugs. For these reasons, we argue that a decentralized structure is a better fit for situations in which contributor uncertainty is high. Thus, when contributor uncertainty is high, we anticipate that decentralized communities will demonstrate more commits than centralized communities. Hence,

H4. As contributor uncertainty increases, FLOSS communities with centralized committal structures will have lower performance gains than communities with decentralized committal structures.

4.5. Contingency factors that increase IP needs: Task interdependence

Task interdependence in software development occurs between developers when the source code files they are working on have functional dependencies that require developers to coordinate [21,59]. FLOSS communities rely on two types of processes to manage interdependence [59]: intra-unit interdependence and inter-unit interdependence [97]. Considered a good software practice, intra-unit interdependence occurs within software modules, where individuals working on similar and related functionality coordinate their activities to complete a task [24]. Such organization allows modules to exhibit a high degree of within-module interdependency, which has been referred to as cohesion [76]. Since similar functionality is contained in single modules, developers working in each module must share similar knowledge in order to collaborate more effectively and a single owner is responsible for coordinating its development [21,24]. Due to shared knowledge and centralization, intra-unit interdependence enables more effective coordination and thereby lowers demand on a FLOSS communities' information processing infrastructure.

Inter-unit interdependence occurs when community members must coordinate activities due to functional dependencies between separate modules [21]. Where intra-unit dependencies address problems localized to a single module, inter-unit dependencies address problems that require altering one or more modules. Because contributors often possess module-specific knowledge, inter-unit coordination is more difficult than intra-unit coordination [38,97]. It forces committers to perform more development-related information-processing tasks and invest less time in creating patches. As a result, inter-unit dependencies result in higher information coordination costs across modules [38], higher learning costs due to the need to build shared knowledge necessary for developing the patch [101], and reduces overall productivity of committers in the community. Due to these coordination and learning costs, inter-unit dependencies effectively negate distributed development structures' benefits limiting developers' ability to specialize in specific modules. In sum, task interdependence is higher when inter-unit independence is higher and intra-unit interdependence is lower.

Due to language and knowledge differences across modules, FLOSS community members have a harder time directly coordinating patches that address inter-unit interdependences. As a result, committers must coordinate collaboration across modules and are more likely to become bottlenecks in information flows and development processes. When inter-unit interdependencies are reduced, there may be additional unintended productivity benefits in FLOSS communities. Because committers are coordinating less, they have more time for information sharing and specialized development activities (Conaldi and Lomi, 2013). As a result, the whole community benefits from committers' increased responsiveness to communication and knowledge sharing [97] and from the specialization that results from modularization [38]. These improvements make the community's overall development more efficient and increase its overall development performance. Therefore,

H5. Task interdependency will be negatively related to a FLOSS community's performance.

Since task interdependence is created from dependencies in the source code [12,21,95], we argue that coordination-related information-processing requirements may be mitigated by the modularity of the software design. Modularity refers to the number of interdependencies across software modules. Limiting cross-module dependencies reduces the need for coordination and communication because contributor's development tasks would be restricted to specific modules [21]. Such a limitation is equivalent to the strategy suggested by Galbraith [33] for creating self-contained tasks. Furthermore, one of the assumptions that Brooks [12] mentions as to what would lead to an increase in the number of communication channels is serialization (i.e., dependencies) between tasks. Removing the serialization constraint would reduce the need to communicate between developers. Collectively then, when the source code itself exhibits few inter-unit interdependencies (e.g., between modules) and more intra-unit interdependencies (e.g., within a single unit), and then there is little need for contributors to coordinate their activities across modules [6,66,83,95]. Modular designs enable developers to work in parallel, and by reducing coordination requirements; it also reduces the coordination-related costs associated with adding more committers [5]. Therefore, we argue that low centralization (e.g., distributed development) is a good fit for communities with more modular codebase (lower inter-unit interdependence).

Specifically, when inter-unit interdependence is high, we anticipate that FLOSS communities with a centralized committal structure and less modular codebase will demonstrate higher performance when compared to communities with decentralized committal structure and less modular codebases. We attribute this higher performance to centralized communities ability to manage coordination-related information-processing requirements that result from less modular source code design. Therefore,

H6. As task interdependency increases, FLOSS communities with centralized committal structures will have higher performance gains than communities with decentralized committal structures.

5. Research method

In order to test our model, we used the source code repositories of 237 projects. We drew our variables by coding data in these repositories. Below, we describe our sampling and data collection procedures, followed by the construct operationalizations and a description of the controls used.

5.1. Sample

Similar to prior FLOSS research [22], we drew a theoretical sample of FLOSS communities. The following logic guided our sampling. First, we limited our sample frame only to projects that were likely to have ongoing development activity [22]. This is important because a large proportion of FLOSS projects are dormant [47]. Absent active development, we would lack the ability to observe variations in performance across communities. Second, we used an established repository to identify FLOSS communities. Specifically, we used ohloh.net,[6] a website that lists over 275,000 FLOSS projects, to identify the population of active FLOSS projects. Third, consistent with Wu et al. [103], we drew data from a subset of the top 1000 listed projects. Fourth, consistent with prior work, we selected a subset of projects that used similar programming languages (e.g. [58,63]). This is important because languages vary in size and complexity ([63]; Wyuker; Jones; Seemidharefs). Therefore, we examined FLOSS communities that are widely diffused and have varying programming philosophies: C, C++, and Python. Using these heuristics, we identified 289 potential projects out of the top 1000 listed projects on ohloh.net for inclusion in our analysis.[7]

5.2. Data collection

Consistent with prior work using source code repositories (e.g. [42, 56,63,103]), we screened the 289 projects and excluded any projects

[6] Now known as the black duck open hub since July 2014, see http://openhub.net.

[7] It is important to note that where prior research sampled from sourceforge.net [87], we drew our sample from FLOSS projects listed on ohloh.net. We did so because ohloh.net represents a broader sample of FLOSS communities than sourceforge.net. When we compared our sample with sourceforge.net, we found that only 22% of the top 1000 projects listed on ohloh.net were hosted on sourceforge.net. Because it draws on a broader sample frame, we expect our results to have more external validity than prior research. Interestingly, shortly after drawing our sample, sourceforge.net purchased ohloh.net to increase its FLOSS-hosting market share [77,78].

that were inactive,[8] umbrella projects hosting multiple projects, or had inaccessible code repositories. As a result, we were left with 237 projects that we sampled our observations from. We then downloaded these projects' source code repositories and coded committal activity in three-month increments. We also extracted names of the unique contributors from the committal logs of the revision control system. As a result, we collected a total of 1832 quarterly observations from 237 projects for activity between Jan 1st, 2007 and June 30th, 2009. Table 3 provides the sample's descriptive statistics.

5.3. Variables

Table 4 lists the variables we collected from this sample and provides a summary of the corresponding construct definitions and operationalization.

5.3.1. Performance

Performance was operationalized as the number of completed commits in a period [40,45]. To estimate performance, we counted the total number of commits made by a project performed during a three month period. Given the wide variance in the number of commits across projects, we performed log transformation to normalize the distribution of the data.

5.3.2. Centralization

Centralization was operationalized as the ratio of committers to the total number of developers (i.e., contributors and committers) in a community. The higher the ratio, the greater the distribution of workload representing more decentralized structures; lower ratios represent more centralized structures. To estimate centralization, we counted the total number of committers and contributors as identified from the revision control system for the whole analysis period. We then calculated the ratio of committers to total number of committers and contributors as an estimate of decentralization of committal structure. We subtracted that ratio from one to create a figure that increased with centralization. Our calculated figure was not normally distributed with distributional masses close to both zero and one. As a result, we transformed the variable into a nominal variable using a median split [57], where zero represents a decentralized committal structure and one represents a centralized committal structure.[9]

5.3.3. Task routineness

Task routineness[10] (simplicity) was operationalized as the total number of source files changed per commit. For software development tasks, simple routines and steps can be implemented and grouped into a single source file. Non-routine development tasks have unique characteristics and draw from different functional modules and require the modification or addition of more than a single source file. Changes to more source files requires developers to acquire the embedded knowledge and increases the risk of bugs or conflict with another developer, as the likelihood of developers working in parallel on the same file increases [21]. Therefore, the average number of files changed per commit will be inversely related to the routineness of the development task. To estimate task routineness, we counted the total number of files changed or added over the analysis period. We then divided that number by the number of commits to estimate the average number of files changed per commit. We then log transformed the variable to normalize its distribution and mean center as recommended by Aiken and West [3] when testing for interaction terms. Since our variable is mean centered, we multiply it by negative one to reverse the direction of the variance. This was necessary because routine tasks are associated with a smaller number of changed or added files.

5.3.4. Contributor uncertainty

Uncertainty in task environment refers to unpredictability tied to integrating work of new or external contributors to a FLOSS project. New contributors are likely to create ramp-up effects either because they know less about the current code base or time required to learn about the development process [12]. As a result, committers find themselves spending more time reviewing and committing the work of first time contributors [101]. We identified the number of unique contributors for the analysis period from the committal logs of the revision control system. We estimated contributor uncertainty using the ratio of new contributors to the total number of committers. However, given the distributional characteristics with masses close to the values of zero and one, we performed a median split and dichotomized the variable into low (zero) and high (one) uncertainty [57].

5.3.5. Task interdependence

Task interdependence refers to the degree to which the completion of development tasks requires developers to cooperate. To estimate the level of dependencies across and within modules, we leveraged the leading eigen-vector method to partition the dependency graph of the source code and obtain an estimate of its modularity [67]. The modularity value is high when cross-module dependencies are low (i.e., loosely

Table 3
Descriptive statistics for the project sample.

	Median	Mean	STD
Age			
In weeks relative to 1-1-2007	4	5.64	48.7
Popularity	57.5	192.1	447.71
Committers count per quarter	5	10.84	14.489
Contributors count per quarter	8	16.29	26.786
Commits count per quarter	116	289.2	456.656

Table 4
Variable operationalization.

′	Definition	Operationalization
Performance (PERF)	Progress made toward meeting the demands of the FLOSS community.	The tallied count of commits listed in the repository within a three month window [40,45].
Centralization (CENT)	The degree to which the committal activity in a FLOSS community is concentrated in the hands of a small group of committers relative to the overall size of the development structure's membership.	The ratio of committers to total number of developers in the community [75].
Task routineness (TROUT)	The degree of predictability in the software development task done within the FLOSS community.	The average number of files changed per commit.
Contributor uncertainty (CUNC)	The unpredictability in development tasks that is introduced from the need to integrate the work from contributors who are considered external to the FLOSS development structure.	The ratio of new contributors to the total number of contributors during the analysis period.
Task interdependence (TINT)	The degree to which the development tasks of the developers in a FLOSS community require cooperation to be completed.	The modularity measure [68,69] of the leading eigen-vector partitioning [67] of the dependency graph for the beginning of the analysis period.

[8] Projects were considered inactive if they didn't have at least a single commit per quarter during the timeframe of our study.
[9] Since this is an important construct in our model, we test the robustness of our results using alternative splits to rule out that our results are an artifact of the median split. We use the median split results because they are the easiest to present and interpret. See Appendix A for details.
[10] Task routineness refers to the predictability of the task. Predictable tasks are analyzable and therefore easy to break down into simple routines, grouped into a single source file. There is clearly overlap between this and the concept of task complexity.

coupled) and within-modules dependencies are high, suggesting that the modules are partitioned with very little interdependence of tasks across modules. First, we extracted the dependency graph for a snapshot of the source code at the beginning of the analysis period. We then examined how well the graph could be partitioned into sub-graphs. Next, we extracted the modularity measure for the resulting partition from the leading eigen-vector method. We then mean centered the variable since we would be testing an interaction term [3]. Finally, we multiplied the value by negative 1 since the modularity estimate was inversely related to task interdependence.

5.4. Controls

5.4.1. Age of project

Age was measured in terms of number of months from the time the first committal was made to the source code up to the first day of the analysis period. FLOSS project may affect the number of committers since older projects have more established roles and procedures for coordination than younger projects. In addition, age can serve as a proxy for the stage in which the FLOSS project is, which could affect the level of activity in the project and, thereby, its performance [92].

5.4.2. Programming language

Because we selected a set of programming languages with varying philosophies, we controlled for the FLOSS project language. This is important because some languages, like Python, put more emphases on productivity and readability rather than performance [63,92]. As a result, we created dummy coded variables to control for programming language in our analysis.

5.4.3. Project popularity

A FLOSS project's popularity may attract developers due to the social benefits associated with participation and exposure [52,55]. In addition, popular projects tend to have a greater number of software users, which means the pool of potential contributors to the project is larger. To measure popularity, we drew the number of users of a project from ohloh.net.

5.4.4. Size of project

Projects with a larger code base are likely to be more complex and require more patches. Research has found that rate of change in the source code is closely related to the Source Lines of Code (SLOC) [11]. Therefore, we control for size of the code base, estimated as SLOC, at the beginning of the analysis period, which allowed us to compare differently sized projects.

5.4.5. Number of committers

While we include the ratio of committers to new contributors in our study, the total number of committers may have implications for FLOSS project performance. Larger projects can perform a greater number of commits simply by virtue of having a larger group of committers. As a result, we controlled for the number of committers, as we are interested in the effects of structure that are independent of FLOSS project size.

6. Analysis and results

We used a mixed-model analysis to evaluate our hypotheses [8,19]. Specifically, we used the lme4 library [7] for GNU R [80] to estimate a random intercept model where observations are nested within projects. Our final model specification involves variables that vary within projects over time (level 1), variables that vary across projects but do not vary over time (level 2), Interaction terms, and multiple error terms.[11]

[11] Model specification can be inferred from Table 6 based on the values included in each column.

Level 1 variables are Centralization (CENT), Task Routineness (TROUT), and Contributor Uncertainty (CUNC). The first step in mixed model analysis is to assess the Interclass Correlation Coefficient (ICC) for our variables. We found our level 1 variables to have ICC values of 0.45, 0.43, and 0.18 respectively. The non-zero ICC values suggest that some of the variability in our data can be explained simply by project membership (i.e., they also have level 2 effects). Econometricians refer to this problem as the endogeneity problem, and to address it, we include the group mean for each of our level 1 variables into our model to assess the level 2 effect for our variables [34]. We denote these level 2 variables with the grp suffix. Task Interdependence (TINT) had an ICC of 0.97, meaning the variability is mostly at level 2 [8]. As a result, it will be treated as a level 2 variable, along with all our control variables that do not vary by time, with the exception of the time effect.

We follow a bottom up approach in building our model [102]. We start without intercept only model that we will be used as the base to compare all our models. We start by including each level 1 effect and check if it has a significant random effect. After completing this step, we identify Time, along with all our level 1 variables to have associated random effects. In addition to the residual error term, all our random effects are assumed to be normally distributed, and have zero mean and non-zero variance.

The next step in our analysis protocol is to include level 2 effects, then level 1 by level 1 interactions, and finally cross level interactions. Using cross level interaction terms, we can assess whether the effects of our predictors change as project membership changes. In longitudinal data analysis, the interaction of time with level 2 variables is assessed to model any time growth effects. We found all the time interaction terms to be insignificant and were excluded from the model to reduce its complexity [8,34].

To validate the assumptions of mixed model analysis, we graphically assessed the distributions of our variables and residual plots and found no signs to the violation of the normality assumptions or homoscedasticity. Furthermore, inter-variable correlations had low magnitudes (see Table 5) suggesting discriminant validity and no violation of the independence assumption. Finally, we obtained a maximum value of 0.0827 for Cook's d suggesting that there are no outliers exerting any undue influence on our results [19,71]. Overall, we did not detect any problematic data points or violations of mixed-model analysis's assumptions and feel confident about the validity of our results.

Table 6 summarizes the results from our mixed-model analysis and gives estimates of the β coefficients of our different models [19]. Inference tests were based on t-tests, where Satterthwate's approximation of degrees of freedom was used to obtain conservative p-value estimates [51].

Effect size was estimated using reduction in residual, intercept, and slope random variation, which we refer to as pseudo R^2 [8,34]. Level 1 effects would result in a reduction on residual variance. Level 2 effects would result in reduction in intercept random variation. Finally, cross level interactions would result in random slope variation. Since the main effect model included both level 1 and level 2 effects, we get two pseudo R^2 components which are 61.7% reduction in residual variance, and 50% reduction in intercept variance.

We found support for all our main effects hypotheses. The null hypotheses that the regression coefficient is different from zero is rejected with an alpha $< .05$ for t-tests. We found that task routineness (TROUT) has a significant relationship with PERF with a coefficient of 0.44 ($p < .001$) in the main-effect model; H1 was supported. Furthermore, the positive 0.2 ($p < .001$) level 2 task routineness coefficient (TROUTgrp), suggests that projects that generally face more routine tasks perform better on average than projects that are faced with more complex tasks, lending further support to H1. Similarly, for H5, there is a negative relationship between task interdependence (TINT) and performance. The results suggest that TINT ha a significant relationship with PERF with a coefficient of -0.91 ($p < .001$).

Table 5
Variable correlations and descriptive statistics.

	PERF	CENT	TROUT	CUNC	TINT	AGE	PER	isGPL	isC	isCpp	isPy	POP	SLOC	COM
PERF	1.00													
CENT	0.22	1.00												
TROUT	0.24	0.03	1.00											
CUNC	0.16	0.33	0.10	1.00										
TINT	-0.27	-0.03	0.21	-0.01	1.00									
AGE	0.06	0.11	0.00	0.02	-0.15	1.00								
PER	-0.06	0.03	-0.02	-0.15	-0.01	0.04	1.00							
isGPL	-0.01	-0.01	0.01	0.02	0.00	0.10	0.05	1.00						
isC	0.21	0.12	0.10	0.02	-0.11	0.34	0.00	0.10	1.00					
isCpp	0.20	-0.06	-0.24	-0.06	-0.29	-0.09	-0.01	0.11	0.03	1.00				
isPy	-0.13	-0.05	0.10	0.01	0.20	-0.29	0.00	-0.15	-0.63	-0.32	1.00			
POP	-0.20	-0.13	-0.01	0.00	0.09	-0.39	-0.03	-0.11	-0.18	0.08	0.18	1.00		
SLOC	0.44	0.14	-0.28	-0.02	-0.44	0.32	0.06	-0.04	0.35	0.29	-0.36	-0.26	1.00	
COM	0.66	-0.08	0.14	0.15	-0.22	0.13	-0.01	0.04	0.19	0.17	-0.07	-0.24	0.39	1.00
min	0.00	0.00	-4.64	0.00	-0.42	-7.61	1.00	0.00	0.00	0.00	0.00	-0.21	-6.01	-1.77
mean	4.68	0.48	0.00	0.50	0.00	0.00	4.54	0.42	0.82	0.43	0.14	0.00	0.00	0.00
median	4.84	0.00	0.01	0.00	0.03	0.27	4.00	0.00	1.00	0.00	0.00	0.00	0.01	0.02
max	8.33	1.00	5.03	1.00	0.65	13.16	10.00	1.00	1.00	1.00	1.00	0.15	3.73	3.05
std	1.64	0.00	0.86	0.00	0.29	2.65	2.97	0.00	0.00	0.00	0.00	0.11	1.41	1.15

For contributor uncertainty, we found that projects that have more new contributors perform worse on average than projects that have less new contributors. This can be inferred from the group level contributor uncertainty coefficient (CUNCgrp) which is −1.02 ($p < .001$), lending support to H3. However, within a project, adding a new contributor is likely to increase performance with a positive CUNC coefficient of 0.18 ($p < .001$), suggesting a relationship opposite to the one we hypothesized in H3. We attribute this to the operationalization of our variables, since adding a new contributor brings with him/her a source code contribution within a project. However, when comparing across projects, the level 2 effect of contributor uncertainty (CUNCgrp) clearly illustrates support for H3 given how projects with less new contributors show higher levels of performance.

Table 6
Model fitting.

	Null	Main effects	Interaction effect
CENT ∗ TINT			0.27 (0.24)
CENT ∗ CUNC			−0.17 (0.08)*
CENT ∗ TROUT			0.27 (0.10)**
TINT		−0.91 (0.25)***	−1.02 (0.27)***
CENT		0.50 (0.06)***	0.59 (0.07)***
CUNC		0.18 (0.04)***	0.25 (0.06)***
TROUT		0.44 (0.07)***	0.34 (0.08)***
CENTgrp		0.78 (0.17)***	0.83 (0.17)***
CUNCgrp		−1.02 (0.20)***	−1.02 (0.20)***
TROUTgrp		0.20 (0.17)	0.22 (0.17)
PER		−0.03 (0.01)**	−0.03 (0.01)**
AGE		−0.08 (0.02)***	−0.07 (0.02)***
isGPL		−0.10 (0.11)	−0.11 (0.11)
isC		−0.12 (0.17)	−0.09 (0.17)
isCPP		0.10 (0.12)	0.12 (0.12)
isPy		−0.47 (0.20)*	−0.45 (0.20)*
POP		0.56 (0.62)	0.58 (0.62)
SLOC		0.01 (0.04)	0.02 (0.04)
COM		1.38 (0.04)***	1.36 (0.04)***
(Intercept)	4.64 (0.09)***	4.56 (0.12)***	4.51 (0.12)***
Num. obs.	1832	1832	1832
Num. groups	235	235	235
Variance			
Residual	0.94	0.36	0.35
Intercept	1.80	0.90	0.90
CENT		0.16	0.12
CUNC		0.05	0.06
TROUT		0.43	0.45
PER		0.01	0.01
Pseudo R^2			
$R^2_{Residual}$	–	61.7%	62.7%
$R^2_{Intercept}$	–	50%	50%
R^2_{CENT}	–	–	25%

S.E. between parentheses.
*** $p < 0.001$.
** $p < 0.01$.
* $p < 0.05$.

We found support for all but one of our moderation hypotheses. The interaction model estimated the interaction between our main effect variables and CENT, which represents the centralization of a FLOSS community's committal structure. The interaction effects model included both level 1 by level 1 interaction and cross level interactions. The level 1 interactions (TROUT and CUNC interactions) reduced the residual variation by only 1%. In Psychology and Management studies, the median effect size for interaction studies over the past 30 years was around 0.002 [1]. McClelland and Judd [62] attribute this small effect size to the small residual variance, after accounting for the main effects, which is used to detect moderation effects, which also might explain the difficulty in detecting a significant effect for our unsupported moderation effect. The cross level interaction of task interdependence reduced random slope (CENT) variation by 25%.

H2, which suggests FLOSS communities with centralized committal structures are a better fit for routine tasks, was supported. Analysis yielded a positive, significant coefficient of the TROUT ∗ CENT interaction (0.27, p-value < 0.01). To understand this interaction's implications, we plotted the simple slopes for the interaction term (see Fig. 3). The plot suggests that as the development task grows more routine, performance increases at a higher rate for centralized committal structures than for decentralized committal structures [3,19]. The simple slope for decentralized FLOSS communities is represented by the main effect coefficient associated with TROUT in the interaction model (0.34, p-value < 0.0001). We found that the TROUT simple slope for the decentralized reference group was significantly different from zero [3,19].

To obtain the TROUT simple slope for the centralized FLOSS communities, we added the TROUT ∗ CENT coefficient to the TROUT coefficient (0.34, p-value < 0.0001). The significance test for this simple slope was obtained by reverse coding the CENT variable such that the centralized FLOSS projects are the reference group and then refitting the interaction model [3,19]. The higher increase in performance is indicated by the steeper and positive slope of the centralized committal structure line (see Fig. 3). Since TROUT is a continuous and mean-centered variable, the significance of the interaction term TROUT ∗ CENT (p-value < 0.05) was used to confirm that the difference in the simple slopes between the two levels of CENT is indeed significant [3].

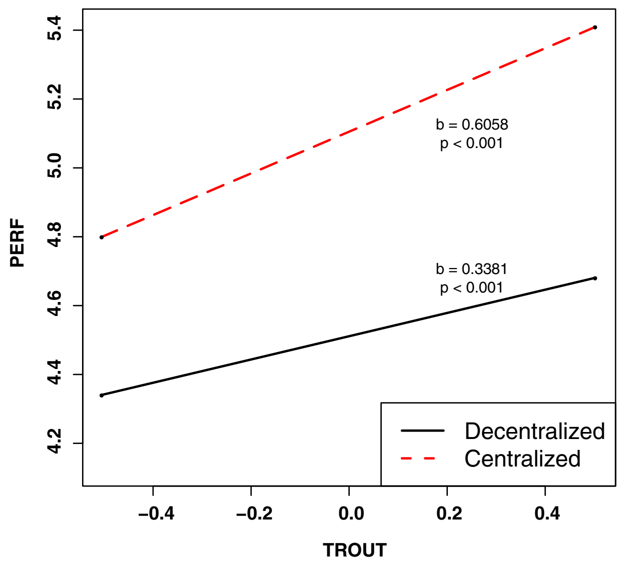


Fig. 3. Simple slopes for effect of TROUT over different levels of CENT.

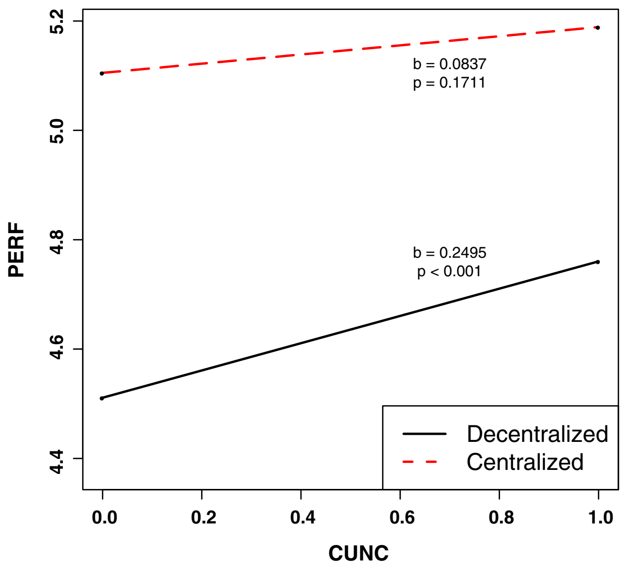


Fig. 4. Simple slopes for effect of CUNC over different levels of CENT.

The negative and significant slope of the CUNC ∗ CENT coefficient (-0.17, p-value < 0.05) lends support to H4, which suggests that a decentralized committal structure is a better fit for dealing with higher contributor uncertainty (i.e., the increase in numbers of new contributors). This result also confirms that the simple slopes for centralized and decentralized FLOSS communities are significantly different [3]. To illustrate this difference, we plotted the simple slopes for the interaction term (See Fig. 4). The graph shows that the effect of uncertainty on performance has a positive and steeper slope for decentralized communities than for centralized communities. The CUNC simple slope for centralized FLOSS communities is non-significant (0.0837, p-value > 0.1),[12] suggesting that contributor uncertainty has no impact on performance for centralized FLOSS communities. On the other hand, the CUNC simple slope for decentralized communities was positive and significant (0.25, p-value < .05), which is contrary to our expectations.

Finally, we found no conclusive support for H6, although the coefficient of the TINT ∗ CENT term (0.27, p-value > 0.1) while non-significant, follows our predicted positive relationship. It is possible that we were unable to detect a significant relationship because only a small portion of residual variance remains after fitting the direct effects of the model, resulting in the moderation effects being hard to detect [62]. If the results were significant, they would suggest that FLOSS communities with centralized committal structures perform better than communities with decentralized committal structures as task interdependence is increased. To illustrate this, we plotted the simple slopes for the interaction term (see Fig. 5). The steeper slope for the relationship between TINT and PERF for decentralized committal structures suggests that centralized committal structures are a better fit for increasing task interdependence, as performance drops at a much slower rate (-0.75, p-value < 0.001) than in decentralized committal structure (-1.02, p-value = <0.0001).[13] Furthermore, when looking at the increasing performance (right to left side of the graph), we observe that decentralized committal structures have higher performance gains as task interdependence is reduced through more modular software design. However, the difference between these lines needs further investigation, as the non-significant interaction coefficient TINT ∗ CENT (0.27, p-value > 0.1) means that there might not be any difference between centralized and decentralized structures in our data [3].

7. Discussion

We used an organization information processing lens to better understand how FLOSS community structure influenced performance. We forwarded the argument that community structure ties to performance varied with the task routineness, community composition, and task interdependence. Overall, we found support for all our hypotheses but one (Table 7) and summarize the main contributions in Table 8.

In our first set of hypotheses, we found that FLOSS communities will perform better when their development tasks are simpler or more routine (H1). We attributed the improved performance to the reduced information-processing requirement of routine tasks, which are less likely to overwhelm the committal structure. This finding suggests that FLOSS communities could improve their performance by simplifying development tasks. Specifically, communities should invest their time in organizing the source code such that tasks requiring similar functionality may be managed by modifying fewer files [76].

In addition, we found support for centralized committal structures (H2) being a better fit for routine development tasks. This is due to lower coordination requirements between community members operating within centralized committal structures when compared to decentralized structures. When tasks are simple or routine, we reasoned that committers will be required to communicate less with other committers as well as will receive fewer requests for help from contributors. Because less communication is required to create patches ready to be committed to the codebase, we believe that community information processing capability will be less likely be overwhelmed and more likely to enable performance. Because we did not directly measure the amount of communication within the community, we believe that this finding suggests a need for research that examines the quantity and quality of communication tied to routine and non-routine FLOSS development. Because such communication is not included in the FLOSS codebase, it may be necessary to employ different research methods (e.g., survey based or qualitative techniques) to investigate the implications of

[12] The significance test was obtained after reverse coding CENT and refitting the interaction model and using the significance test for the CUNC coefficient [3].

[13] The significance test was obtained after reverse coding CENT and refitting the interaction model and using the significance test for the TINT coefficient [3].

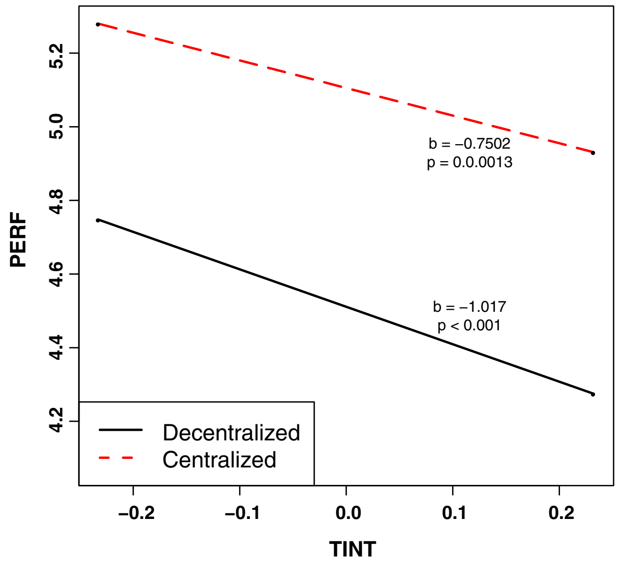


Fig. 5. Simple slopes for effect of TINT over different levels of CENT.

communication, task routineness and community structure on FLOSS performance.

Collectively, these results point to routineness as an important contingency that needs to be matched with centralized and decentralized committal structures in order to enhance FLOSS community performance. Even though FLOSS communities with decentralized committal structures might have a higher capacity to process information, our findings imply that they may not necessarily outperform more centralized communities. In fact, they suggest that FLOSS projects that engage in routine development activities, either due to the simplicity of the problem or the maturity of project, operate effectively with a centralized committal structure. On the other hand, when problems are less routine and require extensive changes to the codebase, our findings suggest that decentralized, distributed committal structures may enable higher FLOSS project performance. However, as we note below in our discussion of H5 and H6, the community structure's implications for performance may be limited by interdependencies in the codebase.

In our second set of hypotheses, we probed the implications of contributor uncertainty and structure for FLOSS performance. We did not find support for our hypothesis that suggested that increased uncertainty attributable to new contributors participate will have a detrimental effect on performance of a project, however we did find evidence that projects with less uncertainty from new contributors tend to perform generally better (H3). Our analysis did not support Brooks' view that new contributors, and their patches, will slow development cycles and increase costs of software. One could speculate that that the dichotomization of the uncertainty variable was the source of our failure to detect effects tied to uncertainty within project [57]. However, as noted in the method, our analysis adhered to best practices and was robust to several post-hoc tests. A plausible alternative explanation for our findings lies in the spirit of FLOSS communities. It could be that when uncertainty overwhelms committal structures, that community members will rally, organize committal activities, and ensure that the FLOSS project continues to advance. To investigate this alternative explanation would require either conducting case study research focused on analyzing the content of communication among committers or collecting data on the number of patches ignored by committers. Future research that examines FLOSS communities' response to new contributors and uncertainty constitutes a rich direction.

Although uncertainty did not directly affect performance within project, we did find a significant negative interaction between centralization and uncertainty. We found that FLOSS communities with decentralized committal structures tend to perform better under higher uncertainty (H4). This is interesting, because our analysis suggests that uncertainty did not affect centralized committal structures' performance. The interaction effect could be attributable to decentralized committal structures being able to process information requests and patches forwarded by new contributors while also

Table 7
Summary of empirical findings from the higher-order model.

Hypothesis	Coefficient	Support
H1: Task routineness is positively related to a FLOSS community's performance.	TROUT: 0.44 p-value < 0.001	Supported
H2: As tasks become more routine, FLOSS communities with centralized committal structures will have higher performance gains than communities with decentralized committal structures.	TROUT*CENT: 0.27 p-value < 0.01	Supported
H3: Contributor uncertainty will be negatively related to a FLOSS community's performance.	CUNCgrp: −1.02 p-value < 0.001 CUNC: 0.18 p-value < 0.001	Supported when comparing projects, not within project
H4: As contributor uncertainty increases, FLOSS communities with centralized committal structures will have lower performance gains than communities with decentralized committal structures.	CUNC ∗ CENT: −0.17 p-value < 0.05	Supported
H5: Task interdependency will be negatively related to a FLOSS community's performance.	TINT: −0.91 p-value < 0.001	Supported
H6: As task interdependency is increased, FLOSS communities with centralized committal structures will have higher performance gains than communities with decentralized committal structures.	TINT ∗ CENT: 0.27 p-value > 0.1	Not support

Table 8
Summary of contributions.

Finding	Impact
FLOSS communities performing simple tasks perform better.	• Importance of good source code design to simplify development. • Contributors should work closer to the community and make small incremental changes rather than work in isolation and accumulate their patches into a single patch that is hard to incorporate.
Centralized committal structures are a better fit for simpler routine tasks.	• There is no single superior structure. • The committal structure should match the needs of the community.
Decentralized committal structures are a better fit under high contributor uncertainty.	• Decentralized committal structures are necessary if community involvement is valued. • Brooks' law is not obsolete; the committal structure has to be overwhelmed for it to become obvious.
Task interdependence increases information-processing requirements for a FLOSS community and reduces performance.	• Importance of modularizing the source code and its effect on the performance of a FLOSS community. • Further validation of the Newman [67] modularity measure.
Decentralized committal structures are a better fit under conditions of low task interdependence.	• Centralized committal structures might be the only way to continue to maintain and develop tightly coupled code bases. • Decentralized committal structures are enabled by modular design. • Brooks' and Raymond's views are complementary. Raymond explains how FLOSS development is conducted under conditions of fit, while Brooks' views become apparent under condition with lack of fit.

continuing to commit patches from existing contributors. Since decentralized committal structures are able to process patches in parallel, the majority of committers may be able to continue committal activity, even as a minority of committers provides the new contributors information and attention necessary to become active members of the FLOSS project. This suggests a positive cycle that is spawned if committers are able to invest in new contributors. In such a case, a FLOSS project may be more like to convert new contributors to regular contributors and increase overall productivity of the community.

What is interesting about this finding is that it directly speaks to the tension between Brooks' and Raymond's views. The lack of performance improvement for centralized structures shows, as Brooks' had anticipated, that there are no performance gains with the addition of new contributors. The performance increase of decentralized structures under conditions of uncertainty suggests, as Raymond's work implies, that decentralized communities may be more apt to become bazaars of ideas, integrate patches offered by new contributors, and yield more commits. In light of these contrasting implications, we believe it may be premature to declare Brooks' law obsolete or irrelevant to FLOSS communities. Rather, it seems that Brooks' law applies to FLOSS communities when the committal structure reaches the limits of its information-processing capacity (e.g., centralized under uncertainty). In such a circumstance, we believe that even if one added a limitless number of contributors one would see a relatively flat level of performance at the project level. In order to address this limitation of centralization, we believe would require FLOSS community organizers to reduce bottlenecks, decentralize, and increase their capacity to process information. However, because we do not investigate how changing FLOSS community structures ties to performance, this implication suggests a need for research that examines how shifts in FLOSS community structure ties to performance.

In our final set of hypotheses, we examined the implications of software's modularity and community structure on performance. To examine these hypotheses, we introduced to the literature an objective measure of modularity that is derived from the actual codebases of FLOSS projects [67]. We found that communities with fewer interdependencies in their development task (e.g., greater modularity in their software) performed better (H5). We attribute this improved performance to the lower coordination costs which frees committers to perform more development or commits to a code-base. The need from coordination stems mainly from dependencies between the source files on which different developers work, which creates the need for coordination Crowston [21]. By improving the design of the source code to exhibit higher modularity (i.e., low coupling between modules and high cohesion within a module), our findings imply that FLOSS communities can improve their performance.

When FLOSS communities do not develop highly modular codebase, we expected that centralized committal structures are more apt to lead to higher performance (H6). From an information processing viewpoint, this is due to the need for greater centralization to better coordinate highly interdependent tasks. In communities with decentralized committal structures and less modularity, committers will have to maintain communication channels with a larger group of committers to maintain the functional integrity of the software [12,21]. With higher task interdependence, the amount of information that needs to be exchanged in each channel will increase, making it impossible for a committer in a decentralized structure to quickly process patches. Therefore, when software modularity is low and interdependence is high, our analysis suggests that centralized structures are more apt to lead to FLOSS community performance. While our results approached significance (e.g., $p > .01$), we did not find conclusive evidence in our data to support this view. One explanation is that interaction effects are very difficult to detect given that the residual variance left from fitting the direct effect is small [62]. Consequently, we believe there is a need for future research that further explores the interplay between modularity and committal structure.

Overall, this finding speaks to the importance of software design and FLOSS community structure. In order to enable Raymond's bazaar of ideas, it is important that FLOSS communities with large committal structures construct software with highly modular design. Adding developers or improving performance comes at an information-processing cost, and FLOSS communities will be able to leverage contributions of new contributors when the committal structure and software's structure makes it possible to do so. Absent appropriate community structure and modularity, the community must make a trade-off to either reduce their information-processing needs or increase information-processing capacity in order to enable performance.

8. Limitations

Although we took great care in the design of our study, it is important to note limitations associated with our method. First, we aggregated our data into three month blocks. In prior work, researchers have used anywhere from a one month to a twelve month block. Although a sensitivity analysis using a subset of the data suggested that our results would be essentially unchanged at a monthly unit analysis, we lacked the data to evaluate whether a one year window more accurately captured reality. We were unable to analyze the full dataset at a month-by-month level due to limited computation resources. As research methods and technologies advance, we believe future research may want to examine the influence of varying time periods on FLOSS community structure and performance.

Furthermore, due to some variables' distribution, we dichotomized them. Such an approach might yield spurious relationships and result in lowering the power of the statistical analysis [57]. However, we ruled out the possibility that our results are spurious by conducting the analysis using alternative methods for factoring the variables and found the results to hold (see Appendix A). We used the results of the dichotomized variables because they are easier to present and interpret. The fact that we found significant results despite the lower statistical power of our chosen method suggests that the actual effect sizes could be larger.

Finally, it is important to note that our analysis was based on archival data drawn from actual FLOSS community codebases. As a result, our ability to operationalize theoretical constructs was limited to features of the data archives. Although we took great care to map our theoretical variables to the data, we believe that critics could question the proxy nature of our objective measures and their precise mapping to the constructs. For example, the ratio based centralization variable might not capture all the complexity associated with the construct. There is a possibility that some FLOSS projects that had no contributors might seem highly decentralized because the committers are the only contributors, as in some tightly controlled FLOSS projects. Some quick analysis of our data shows that only 161 observations out of 1832 had no contributors. Excluding these observations from the analysis yielded similar results. While the use of such measures is very challenging, we believe that it is still useful in advancing research in this area. Future research can supplement our analysis of archival data with survey or qualitative data.

9. Future research

Although our work has important implications for the committal structure of FLOSS communities, care should be taken not to infer from our analysis how to structure communication structures within FLOSS communities [23]. There is evidence that a FLOSS community's communication structure need not match its development structure [64]. It would be interesting to examine how the committal (development structure) and communication structure interrelate. For example, future studies could examine whether an increase in communication activity signal that there is a problem in coordination that could be remedied by increased communication; and will this affect the sustainability of the community since it would result in an increased cost for participation and an information overload on the members [43,50]?

Where prior work that examined development organization in FLOSS communities took a small-sample approach [64,74,89], social network approach [16] or mainly examined the communication structure of FLOSS communities [23], this research adopted a more generalizable approach to examining FLOSS committal structures and their implications. We did so as a response to Koch's [44] call for more studies that are generalizable to a diverse set of FLOSS projects. Future research could leverage our approach to sampling, estimating modularity, and mining archival data to take a more granular approach to classifying FLOSS community structures, languages, and application types as a means to understand how these FLOSS community features affect participation and productivity. We also acknowledge opportunities to improve upon our dependent variable. While commits are important to outcomes, it might be conceived as being too narrow. A broader measure like the number of downloads or the number of new users attracted, could enhance this stream of work.

Finally, we only examined a subset of the potential tradeoffs between two committal structures. Future research should examine the implications of different forms of organizing or changing structures for FLOSS communities. For example, what are the actual organizational costs for a community to change the committal structure and how might that change impact contributors and performance over the transition period? It is important to understand what factors might contribute to the success or failure of structural change within FLOSS communities.

10. Conclusion

This study was motivated by a desire to understand how FLOSS communities' structure influences their ability to develop software. Drawing on OIPT [33], we conceptualized FLOSS communities as information processing organizations and identified key contingencies that influence their ability to commit patches to their codebase. Based on our analysis of 237 FLOSS communities, we found that our contingency conditions (e.g., task routineness, uncertainty, and task interdependency) play an important role in determining the optimal committal structure for a FLOSS community. FLOSS community organizers should take these factors into consideration when assigning commit privileges in order to improve FLOSS community performance. More importantly, we consolidate the views forwarded by Brooks [12] and Raymond [81] about software development. While Raymond [81] posited that task independence is a key assumption for the bazaar model to thrive, we provide empirical evidence to support this idea, and identify other factors that come into play. In fact, under certain conditions, we demonstrate that FLOSS communities are similar to traditional software development teams and that Brooks' law [12], as one of the most important classical theories on software project management, still holds true in the FLOSS context.

Appendix A. Robustness of median split results

To ensure that our results are not caused by the median split [57], we performed a tertile split on both CUNC and CENT and refit our interaction model. The results are summarized in Tables A and B below, which show that our results hold; suggesting that the median split did not have an impact on our results. With all continuous variables mean centered, we could interpret the interaction effects and lower order main effects [3].

In Table 7, we found support for H1, H2, H4, and H5. In Table A, we can see that TROUT has a positive and significant coefficient, thus supporting H1. TINT also has a negative and significant coefficient, providing support for H5. In Table A, we have the results from the interaction model. The TROUT $*$ CENT$_{hi}$ coefficient is positive and significant suggesting that the TROUT coefficient for the high centralization group is higher and significantly different than the coefficient for the reference low centralization group, thus providing support for H2. The CUNC$_{hi}$ $*$ CENT coefficients show limited significance and are significant and negative, suggesting that groups with high centralization have a significantly lower coefficient than the reference group with low centralization, providing some support for H4.

Table A
Main-effect model with tertile split of CENT and CUNC.

Term	Coefficient	P-value	Supports
TROUT	0.287	0.0001***	H1
CUNC_med	0.13	0.296	
CUNC$_{hi}$	0.103	0.575	
TINT	−0.43	0.0001***	H5
CENT$_{med}$	0.336	0.0001***	
CENT$_{hi}$	0.857	0.0001***	

*** $p < 0.001$.
** $p < 0.01$.
* $p < 0.05$.

Table B
Interaction model with tertile split of CENT and CUNC.

Term	Coefficient	P-Value	Supports
TROUT $*$ CENT$_{med}$	−0.018	0.315	
TROUT $*$ CENT$_{hi}$	0.11	0.039*	H2
CUNC$_{med}$ $*$ CENT$_{med}$	−0.129	0.315	
CUNC$_{med}$ $*$ CENT$_{hi}$	0.164	0.513	
CUNC$_{hi}$ $*$ CENT$_{med}$	−0.336	0.002**	H4
CUNC$_{hi}$ $*$ CENT$_{hi}$	−0.106	0.096	H4
TINT $*$ CENT$_{med}$	0.347	0.301	
TINT $*$ CENT$_{hi}$	0.241	0.844	
TROUT	0.264	0.0001***	
CUNC_med	0.097	0.75	
CUNC$_{hi}$	0.259	0.006**	
TINT	−0.642	0.001***	
CENT$_{med}$	0.499	0.0001***	
CENT$_{hi}$	0.796	0.0001***	

*** $p < 0.001$.
** $p < 0.01$.
* $p < 0.05$.

References

[1] H. Aguinis, J.C. Beaty, R.J. Boik, C.A. Pierce, Effect size and power in assessing moderating effects of categorical variables using multiple regression: a 30-year review, Journal of Applied Psychology 90 (1) (2005) 94–107.
[2] M.K. Ahuja, K.M. Carley, Network structure in virtual organizations, Organization Science 10 (6) (1999) 741–757.
[3] L.S. Aiken, S.G. West, Multiple regression: testing and interpreting interactions, Sage Pub., Inc., 1991
[4] M. AlMarzouq, L. Zheng, G. Rong, V. Grover, Open source: concepts, benefits, and challenges, Communications of AIS 2005 (16) (2005) 756–784.
[5] C.Y. Baldwin, K.B. Clark, Design rules, The power of modularity, vol. 1, The MIT Press, 2000.
[6] C.Y. Baldwin, K.B. Clark, The architecture of participation: does code architecture mitigate free riding in the open source development model? Management Science 52 (7) (2006) 1116–1127.
[7] D. Bates, M. Maechler, Linear mixed-effects models using S4 classes, R package version 0.999375-312009.
[8] R. Bickel, Multilevel analysis for applied research: it's just regression! Guilford Press, 2007.
[9] P.M. Blau, The hierarchy of authority in organizations, American Journal of Sociology 73 (4) (1968) 453–467.
[10] A. Bonaccorsi, C. Rossi, Why open source software can succeed, Research Policy 32 (7) (2003) 1243–1258.
[11] G. Booch, Measuring architectural complexity, IEEE Software 25 (4) (2008) 14–15.
[12] F. Brooks, The mythical man-month, Proceedings of the international conference on reliable software, vol. 10, ACM Press, 1975.
[13] A. Capiluppi, P.J. Adams, Reassessing Brooks' law for the free software community, in: C. Boldyreff, K. Crowston, B. Lundell, A.I. Wasserman (Eds.), OSS, volume 299 of IFIP, Springer 2009, pp. 274–283.
[16] S. Chou, M. He, The factors that affect the performance of open source software development — the perspective of social capital and expertise integration, Information Systems Journal 21 (2) (2010) 195–219.
[18] J. Colazo, Y. Fang, Impact of license choice on open source software development activity, Journal of the American Society for Information Science and Technology 60 (5) (2009) 997–1011.

[19] J. Cohen, P. Cohen, S. West, L. Aiken, Applied multiple regression/correlation analysis for the behavioral sciences, third edition Lawrence Erlbaum, 2003.
[20] K. Crowston, K. Wei, J. Howison, A. Wiggins, Free/Libre open-source software development: what we know and what we do not know, ACM Computing Surveys (CSUR), 44.2 2012, p. 7.
[21] K. Crowston, A coordination theory approach to organizational process design, Organization Science 8 (2) (1997) 157–175.
[22] K. Crowston, H. Annabi, J. Howison, Defining open source software project success, Proceedings of the 24th International Conference on Information Systems (ICIS 2003) 2003, pp. 327–340.
[23] K. Crowston, J. Howison, The social structure of free and open source software development, http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/view/1207 2005.
[24] K. Crowston, K. Wei, Q. Li, U. Eseryel, J. Howison, Coordination of free/libre open source software development, ICIS 2005 Proceedings, 2005.
[27] W.H. DeLone, E.R. McLean, Information systems success: the quest for the dependent variable, Information Systems Research 3 (1) (1992) 60–95.
[28] J.A. Espinosa, S.A. Slaughter, R.E. Kraut, J.D. Herbsleb, Familiarity, complexity, and team performance in geographically distributed software development, Organization Science 18 (4) (2007) 613–630.
[29] Y. Fan, D. Neufeld, Understanding sustained participation in open source software projects, Journal of Management Information Systems 25 (4) (2009) 9–50.
[30] R.G. Fichman, C.F. Kemerer, The assimilation of software process innovations: an organizational learning perspective, Management Science 43 (10) (1997) 1345–1363.
[31] B. Fitzgerald, The transformation of open source software, MIS Quarterly 30 (3) (2006) 587–598.
[33] J.R. Galbraith, Designing complex organizations, Addison–Wesley series on organization development1973.
[34] A. Gelman, J. Hill, Data analysis using regression and multilevel/hierarchical models, Cambridge University Press, 2006.
[35] A. Gemino, B.H. Reich, R.C. Sauer, A temporal model of information technology project performance, Journal of Management Information Systems 24 (3) (2007) 9–44.
[36] J.F. George, J.L. King, Examining the computing and centralization debate, Communications of the ACM 34 (7) (1991) 62–72.
[37] E.M. Goldratt, J. Cox, The goal, second edition North River Press, 1994.
[38] R.M. Grant, Prospering in dynamically-competitive environments: organizational capability as knowledge integration, Organization Science 7 (4) (1996) 375–387.
[40] R. Grewal, G.L. Lilien, G. Mallapragada, Location, location, location: how network embeddedness affects project success in open source systems, Management Science 52 (7) (2006) 1043–1056.
[41] J. Hage, M. Aiken, Relationship of centralization to other structural properties, Administrative Science Quarterly 12 (1) (1967) 72–92.
[42] J. Howison, K. Crowston, The perils and pitfalls of mining sourceforge, Proceedings of the International Workshop on Mining Software Repositories (MSR 2004) 2004, pp. 7–11.
[43] Q. Jones, G. Ravid, S. Rafaeli, Information overload and the message dynamics of online interaction spaces: a theoretical model and empirical exploration, Information Systems Research 15 (2) (2004) 194–210.
[44] S. Koch, Profiling an open source project ecology and its programmers, Electronic Markets 14 (2) (2004) 77–88.
[45] S. Koch, G. Schneider, Effort, co-operation and co-ordination in an open source software project: gnome, Information Systems Journal 12 (1) (2002) 27–42.
[47] S. Krishnamurthy, Cave or community? An empirical examination of 100 mature open source projects, http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/view/960/881 2002.
[48] G. Kroah-Hartman, J. Corbet, A. McPherson, Linux Kernel development (April 2008), https://www.linuxfoundation.org/publications/linuxKerneldevelopment.php 2008.
[49] G.V. Krogh, E.V. Hippel, The promise of research on open source software, Management Science 52 (7) (2006) 975–983.
[50] G. Kuk, Strategic interaction and knowledge sharing in the KDE developer mailing list, Management Science 52 (7) (2006) 1031–1042.
[51] A. Kuznetsova, P.B. Brockhoff, R.H.B. Christensen, lmerTest: tests for random and fixed effects for linear mixed effect models (lmer objects of lme4 package), R package version, 2(6), 2013.
[52] K.R. Lakhani, R.G. Wolf, Why hackers do what they do: understanding motivation and effort in free/open source software projects, The MIT Press, 2007 3–22.
[53] P.R. Lawrence, J.W. Lorsch, Differentiation and integration in complex organizations, Administrative Science Quarterly 12 (1) (1967) 1–47.
[54] G.K. Lee, R.E. Cole, From a firm-based to a community-based model of knowledge creation: the case of the linux Kernel development, Organization Science 14 (6) (2003) 633–649.
[55] J. Lerner, J. Tirole, Some simple economics of open source, Journal of Industrial Economics 50 (2) (2002) 197.
[56] X. Liu, B. Iyer, Design architecture, developer networks and performance of open source software projects, ICIS 2007 Proceedings, 2007.
[57] R.C. MacCallum, S. Zhang, K.J. Preacher, D.D. Rucker, On the practice of dichotomization of quantitative variables, Psychological Methods 7 (1) (2002) 19–40.
[58] A. MacCormack, J. Rusnak, C.Y. Baldwin, Exploring the structure of complex software designs: an empirical study of open source and proprietary code, Management Science 52 (7) (2006) 1015–1030.
[59] T.W. Malone, K. Crowston, The interdisciplinary study of coordination, ACM Computing Surveys 26 (1) (1994) 87–119.
[60] J.G. March, H.A. Simon, Organizations, Blackwell Publishers, 1993.
[61] M.L. Markus, D. Robey, Information technology and organizational change: causal structure in theory and research, Management Science 34 (5) (1988) 583–598.
[62] G.H. McClelland, C.M. Judd, Statistical difficulties of detecting interactions and moderator effects, Psychological Bulletin 114 (2) (1993) 376–390.
[63] V. Midha, Does complexity matter? The impact of change in structural complexity on software maintenance and new developers' contributions in open source software, ICIS 2008 Proceedings, 2008.
[64] A. Mockus, R.T. Fielding, J.D. Herbsleb, Two case studies of open source software development: Apache and Mozilla, ACM Transactions on Software Engineering and Methodology 11 (3) (2002) 309–346.
[66] N. Ning, S. Kumar, Joint effect of team structure and software architecture in open source software development, IEEE Transactions on Engineering Management 60 (3) (2011) 592–603.
[67] M.E.J. Newman, Finding community structure in networks using the eigenvectors of matrices, Physical Review E 74 (2006) 036104.
[68] M.E.J. Newman, Modularity and community structure in networks, PNAS 103 (2006) 8577.
[69] M.E.J. Newman, M. Girvan, Finding and evaluating community structure in networks, Physical Review E 69 (2004) 026113.
[70] S. Nidumolu, The effect of coordination and uncertainty on software project performance: residual performance risk as an intervening variable, Information Systems Research 6 (3) (1995) 191–219.
[71] R. Nieuwenhuis, B. Pelzer, M. te Grotenhuis, influence.ME: tools for detecting influential data in mixed effects models, R package version 0.72009.
[72] W. Oh, S. Jeon, Membership herding and network stability in the open source community: the ising perspective, Management Science 53 (7) (2007) 1086–1101.
[73] S. O'Mahony, B.A. Bechky, Boundary organizations: enabling collaboration among unexpected allies, Administrative Science Quarterly 53 (3) (2008) 422–459.
[74] S. O'Mahony, F. Ferraro, The emergence of governance in an open source community, Academy of Management Journal 50 (5) (2007) 1079–1106.
[75] W. Ouchi, J.B. Dowling, Defining the span of control, Administrative Science Quarterly 19 (3) (1974) 357–365.
[76] M. Page-Jones, Cohesion, The practical guide to structured systems design, 1998 (Retrieved October 6, 2008, from http://www.waysys.com/ws_content_bl_pgssd_ch06.html).
[77] R. Paul, SourceForge adds support for new version control systems, http://arstechnica.com/open-source/news/2009/03/sourceforge-adds-support-for-new-version-control-systems.ars 2009.
[78] R. Paul, SourceForge wants to be collaboration powerhouse, buys ohloh, http://arstechnica.com/open-source/news/2009/05/sourceforge-acquires-foss-code-metric-web-site-ohloh.ars 2009.
[79] C. Perrow, A framework for the comparative analysis of organizations, American Sociological Review 32 (2) (1967) 194–208.
[80] R Development Core Team, R: a language and environment for statistical computing, R Foundation for Statistical Computing, Vienna, Austria, 2009, ISBN 3-900051-07-0.
[81] E. Raymond, The cathedral and the bazaar: musings on Linux and open source by an accidental revolutionary, revised edition O'Reilly, Cambridge, MA, 2001.
[82] D. Riehle, The economic motivation of open source software: stakeholder perspectives, Computer 40 (4) (2007) 25–32.
[83] R. Sanchez, J. Mahoney, Modularity, flexibility, and knowledge management in product and organization design, Strategic Management Journal 17 (76) (1996) 63.
[84] W. Scacchi, Understanding the requirements for developing open source software systems, Software, IEE Proceedings 149 (1) (2002) 24–39.
[85] C.M. Schweik, R.C. English, M. Kitsing, S. Haire, Brooks' versus Linus' law: an empirical test of open source projects, dg.o '08: Proceedings of the 2008 international conference on digital government research, Digital Government Society of North America 2008, pp. 423–424.
[86] R. Sen, A strategic analysis of competition between open source and proprietary software, Journal of Management Information Systems 24 (1) (2007) 233–257.
[87] R. Sen, S. Chandrasekar, M.L. Nelson, Determinants of the choice of open source software license, Journal of Management Information Systems 25 (3) (2008) 207–239.
[88] P. Setia, B. Rajagopalan, V. Sambamurthy, R. Calantone, A theory and empirical test of peripheral developer contributions to open source development model, Information Systems Research 23 (1) (2012) 144–163.
[89] S.K. Shah, Motivation, governance, and the viability of hybrid forms in open source software development, Management Science 52 (7) (2006) 1000–1014.
[92] K.J. Stewart, A.P. Ammeter, L.M. Maruping, Impacts of license choice and organizational sponsorship on user interest and development activity in open source software projects, Information Systems Research 17 (2) (2006) 126–144.
[95] A. Tiwana, Does interfirm modularity complement ignorance? A field study of software outsourcing alliances, Strategic Management Journal 29 (11) (2008) 1252 (1241).
[96] K.L. Turner, M.V. Makhija, The role of organizational controls in managing knowledge, Academy of Management Review 31 (1) (2006) 197–217.
[97] M.L. Tushman, Work characteristics and subunit communication structure: a contingency analysis, Administrative Science Quarterly 24 (1) (1979) 82–98.
[98] M.L. Tushman, D.A. Nadler, Information processing as an integrating concept in organizational design, Academy of Management Review 3 (3) (1978) 613–624.
[100] Padmal Vitharana, Julie King, Helena Shih Chapman, Impact of internal open source development on reuse: participatory reuse in action, Journal of Management Information Systems 27 (2) (2010) 277–304.
[101] G. von Krogh, S. Spaeth, K.R. Lakhani, Community, joining, and specialization in open source software innovation: a case study, Research Policy 32 (7) (2003).
[102] B.T. West, K.B. Welch, A.T. Galecki, Linear mixed models: a practical guide using statistical software, CRC Press, 2014.

[103] J. Wu, K.-Y. Goh, Q. Tang, Investigating success of open source software projects: a social network perspective, ICIS 2007 Proceedings, 2007.
[104] Chorng-Guang Wu, James H. Gerlach, Clifford E. Young, Examining the determinants of effort among open source software volunteer developers, International Journal of Information and Decision Sciences 5 (2) (2013) 117–139.
[105] U. Zander, B. Kogut, Knowledge and the speed of the transfer and imitation of organizational capabilities: an empirical test, Organization Science 6 (1) (1995) 76–92.

Mohammad Al Marzouq, is an Assistant Professor of Information Systems at Kuwait University.

Varun Grover is the William S. Lee (Duke Energy) Distinguished Professor of Information Systems at Clemson University.

Jason Thatcher is a Professor of Information Systems at Clemson University.