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/

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].

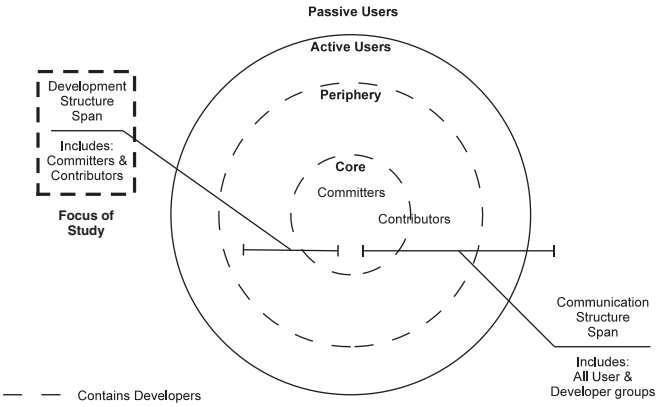

Fig. 1. FLOSS community structure (adapted from [23]).

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.