R&D MANAGEMENT

Task-based structures in open source software: revisiting the onion model

Jose Christian[1,*] and Anh N. Vu[2,*]

[1]CENTRIM, Brighton Business School, University of Brighton, Brighton, UK. j.christian@brighton.ac.uk
[2]University of Sussex Business School, University of Sussex, Brighton, UK. A.Vu@sussex.ac.uk

Studies on Open Source Software (OSS) developer communities have long stated that there is a relationship between community structure and tasks carried out by project members. This relationship has been exemplified by the onion model, which has been instrumental in understanding self-coordination in OSS projects. Despite its ubiquity, there is a lack of empirical evidence to validate the relative position of each task cluster within the onion model. In this study, we map out the community structure of a large open source project and observe its bug-fixing patterns to explore the relationship between tasks and structure. Our study makes three significant contributions. First, we find no empirical evidence to support the structural location of bug-fixing tasks in the onion structure. Second, we find empirical evidence to support the core-periphery continuum model linking an actor's *coreness* to problem-solving ability. Third, our results suggest that the importance and location of each task within the core-periphery structure evolve over time. These findings add clarity to the community structure and their implications for the management and coordination of collaborative innovation projects.

1. Introduction

A key strength of the Open Source Software (OSS) development approach is its large group of volunteer contributors. An OSS community can provide project owners with new ideas, access to resources and help with the dissemination and adoption of the software (Chu and Chan, 2009). Since these communities are made up of an informal volunteer workforce, one of the key challenges remains its management and coordination. To address these challenges, it is important to know more about the structure of the community to understand the key social processes that make OSS development possible (Crowston and Howison, 2005).

From the early academic studies on OSS communities, a persistent link has been made between tasks and observed patterns of behaviour from its contributors. Empirical studies have observed significant differences in the level of commitment and programming skills from community members leading to a de facto core-periphery structure (Wei et al., 2017). Specific software development activities have been associated with each subgroup, placing important managerial tasks at the core and supporting tasks such as bug fixing at the periphery (Rullani and Haefliger, 2013). Due to the duality of roles, project success is contingent on the strategic management of the core and periphery subgroups (Hossain and Zhu, 2009; Rullani and Haefliger, 2013; Daniel and Stewart, 2016). To function effectively, OSS projects require an identifiable core and an active periphery working around the core's activities (Singh et al., 2007). The

Table 1. Approaches to the study of open source

	Interaction	Code development
Social group construct	Social network, user community	Community of practice, team, technical community
Information space	Mailing list or forum	Repository
Scope of tasks	Related to governance and management	Related to Software development (such as bug fixer or active developer)
Centrality attainment	Repeated interaction in discussions	Quality and quantity of technical contributions
Representative studies	Lakhani and von Hippel (2003), Glass (2003), Lehmann (2004), Monteiro et al. (2004), Crowston and Howison (2005), Lin (2005)	Fielding (1999), Franck and Jungwirth (2003), Yamauchi et al. (2000), Nakakoji et al. (2002), Lee and Cole (2003), O'Mahony and Ferraro (2007)

Adapted from Christian (2016).

function of OSS community structure models is to understand the characteristics of the core-periphery subgroups and their role in the development process.

The onion model is a widely used task-based structure emphasising differences in participation, skill and importance between subgroups conducting specific tasks. Through the onion model, studies have been able to explain and understand community dynamics in OSS development (Crowston and Howison, 2005; Jensen and Scacchi, 2007; Jergensen et al., 2011; Nakakoji et al., 2002; Wei et al., 2017). Empirical studies have used the onion model as a framework for identifying developers' progression within and between open source communities (Jensen and Scacchi, 2007; Jergensen et al., 2011). This progression has helped to explain how individuals join projects and specialise in tasks, shedding light on our understanding of coordination and governance in self-organising communities (Madey et al., 2004; de Laat, 2007; Dahlander and O'Mahony, 2011). These studies consider the onion model as a symbolic representation of the decentralised Bazaar style of governance proposed by Raymond (1998) where each 'layer' of the onion represents both a group of individuals carrying out a given task and the importance of each task in the development process.

Despite the onion model's ubiquity in OSS community studies, there is a lack of empirical evidence to validate the position of each task cluster within the onion model. What has yet to be identified, is whether there is a relationship between community structure and tasks. This study seeks to address this gap by answering the following three research questions: (1) Do peripheral developers specialise in bug fixing? (2) Is there a relationship between centrality and bug fixing? and (3) Do tasks-based structures change over time?

To address these questions, we map out the community structure of the LLVM open source project and observe its bug-fixing patterns to identify the relationship between structure and tasks. First, unlike the onion model's implied boundaries between task clusters, we find that developers, regardless of their position within the community, engage in bug fixing activities. Second, our results validate previous studies that found a relationship between an individual's centrality and their ability to carry out problem-solving tasks. Third, we observed a change in bug-fixing patterns over time, suggesting that task-related structures and patterns of contributions evolve over a project's life cycle.

2. Conceptual background and hypothesis development

While the underlying principle of the onion model is based on the concept of a core-periphery structure, it is important to first be clear about its definition and scope. There are two dominant approaches for identifying core-periphery structures in OSS communities, one that focuses on discussions around the management of the project, and the second that focuses on technical contributions made to the project (see Table 1). These approaches emerge from the sources of data utilised, mailing lists and repositories, and often result in different sets of tasks (Sack et al., 2006; Crowston et al., 2012; Christian, 2016). When obtaining data from discussion spaces, such as mailing lists and forums, the emphasis is on managerial and coordination tasks. On the contrary for obtaining data from technical spaces, such as repositories and bug trackers, the emphasis is on software development tasks.

The dominant approach in the study of OSS communities is to focus on technical contributions, with the rationale being that without software production there would be nothing to study (Crowston and

Howison, 2005; Crowston et al., 2012). Through this approach, an actor's position in the core-periphery structure is defined by both quantity and quality of technical contributions made to the project (Fielding, 1999; Yamauchi et al., 2000). OSS communities consist of a small group of highly skilled programmers at its core and a much larger group of average programmers at its periphery (Cox, 1998; Raymond, 1998; von Krogh et al., 2003). The difference in skill levels can lead to task specialisation, where those in the core will typically carry out tasks requiring higher levels of skill, while the majority will contribute through low-skilled tasks (Lakhani and von Hippel, 2003). This core-periphery structure has been instrumental in explaining self-coordination in OSS projects, where 80% of activities are carried out by a small number of individuals (Mockus et al., 2002), therefore, reducing coordination cost (Crowston et al., 2006).

In this section, we discuss three concepts that can help explain the relationship between tasks and community structure in OSS development. Frist, we discuss the intuitive onion-like structure widely used in the open source literature, where the community is divided into task-related groups with different levels of commitment, skills and importance. We then discuss the core-periphery as a continuum where an actor's level of centrality determines their ability to carry out specific tasks. Finally, we look at the evolution of tasks in relation to a project's life cycle. From these areas we develop three hypotheses.

2.1. The onion model and OSS tasks

The onion model remains the dominant interpretation of the link between task and community structure in open source development (Jergensen et al., 2011). Nakakoji et al. (2002) first created an onion-like structure by placing eight development tasks within the core-periphery structure. The onion model has since been extended to include a wider range of tasks such as documentation (Rullani and Haefliger, 2013), design (Bach and Twidale, 2010) and infrastructure maintenance (Christian, 2016). The list of tasks are often identified at the project level and include tasks from both discussion and implementation spaces (Barcellini et al., 2014). The order in which the tasks are arranged is based on a number of factors such as technical complexity of the task (Lakhani and von Hippel, 2003), relative size of the group carrying out that task (Mockus et al., 2002) and a group's influence in the project (Crowston and Howison, 2006; Kilamo et al., 2012). Table 2 provides a list of these activities, their implied structural position, and their space of action. Core-periphery clusters, and their related tasks, are often regarded as mutually exclusive due to their distinct levels of commitment, technical skill, knowledge and access to resources (Mockus et al., 2002; Nakakoji et al., 2002; Lee and Cole, 2003; Jergensen et al., 2011; Scacchi, 2004). Unlike those listed in Table 1, this list includes activities from both information and implementation areas.

When focusing on technical contributions, the onion model is useful for understanding how OSS development communities function (Nakakoji et al., 2002; Crowston et al., 2006; Jergensen et al., 2011; Kilamo et al., 2012). The onion model provides a useful road map for a developers' progression from the peripheral contributor to core member as they get recognition within the community (Ye and Kishida, 2003; Kilamo et al., 2012). The onion model places managerial roles at the core of the structure, showing that coordination of a large volunteer workforce is achieved by centralising decision-making (Scacchi, 2004). The bug fixing process in particular uses the onion model to exemplify the required coordination between most development tasks in order to succeed (Crowston and Scozzi, 2008). It is these tasks and processes that set it apart from traditional closed source development, and therefore, help highlight the key strengths of the OSS development model.

Through the use of the onion model, empirical studies have observed co-dependency between tasks, where core and peripheral groups have complementary roles (Cox, 1998; Raymond, 1998; Rullani and Haefliger, 2013). For instance, while those in the core have both the skill and influence to implement new software features, they rely on a large diverse group of peripheral developers for minor improvements to the software (Crowston and Howison, 2005; Crowston et al., 2006; Jensen and Scacchi, 2007; Jergensen et al., 2011; Wei et al., 2017). From the early literature on OSS development, studies have consistently associated peripheral contributors with bug-fixing tasks for a number of reasons, from leveraging economies of scale and scope (Hienerth et al., 2014) to the relatively low complexity of solutions (Lakhani and von Hippel, 2003). This leads to our first hypothesis:

H1: *Peripheral developers contribute more bug fixes than core developers.*

This first hypothesis is developed based on the established relevant literature and may appear intuitive without additional information. Nevertheless, it constitutes an important part of our study, a main aim of which is to provide empirical evidence on the theoretical literature. To the best of our knowledge, our study is the first to do so.

Table 2. Task-oriented roles in OSS development (Nakakoji et al., 2002; Jensen and Scacchi, 2007; Jergensen et al., 2011; Barcellini et al., 2014; Christian, 2016)

Structural position	Role	Description	Primary area of interaction
Core	Project leader	Includes project owners, initiators, community administrators and project managers	Private communication channels
	Core member	Guiding and coordinating development efforts	Private communication channels
	Active Developers	Regular contributors to the project through new features and bug fixes	Repository
Periphery	Peripheral developers	Casual contributors to the project through new features and bug fixes	Repository
	Bug fixers	Specialised in bug fixes	Repository and bug tracking tools
	Bug reporters	Specialised in reporting bug fixes	Bug tracking tools
	Active users	Users of software who may contribute through low-skilled activities such as documentation and user-to-user assistance	Social media, official and unofficial forums and wikis
	Passive users	Users of the software who do not contribute to the project	Reading list, blogs, social media, official project website

2.2. The core-periphery continuum

A second approach to the study of task-based community structure utilises social network theory to measure the relationship between centrality and problem-solving ability. As with the onion model, the social network approach acknowledges the division of labour between the core and periphery, highlighting differences in influence and access to information needed to carry out specific tasks. For instance, core actors have the highest level of influence in a network and can mobilise resources (Uzzi, 1997; Wasserman and Faust, 1999; Perry-Smith and Shalley, 2003), but often exhibit low levels of information heterophily which limits their ability to solve problems (Granovetter, 1973; McPherson et al., 2001; Burt, 2004; Schilling, 2005; Hargadon, 2006). In contrast, peripheral actors have access to external sources of information and are more likely to introduce novel ideas (Becker, 1970; Perry-Smith and Shalley, 2003; Lakhani, 2006) but lack the influence to implement these solutions (Cattani and Ferriani, 2008; Dahlander et al., 2008). The differences in characteristics create a duality of core and peripheral roles, where the core relies on the periphery for novel ideas and the periphery relies on the core for implementation.

With the duality of the core and periphery being critical for OSS project success and survival, it is important to distinguish one form the other to manage each group accordingly (Crowston et al., 2006; Rullani and Haefliger, 2013). A key challenge emerges, however, when trying to empirically identify where one group ends and the other begins. Citing similar measurement limitations in network theory, Borgatti and Everett (1999) propose a continuous model which sees the core-periphery structure as a continuum rather than a dichotomy. In their continuous model, each actor is given a 'coreness' level between 0 and 1 as a measurement of a node's proximity to the core, with 1 being the closest and 0 being furthest away.

Utilising the continuous model, studies have explored the relationship between coreness and problem-solving ability in online communities. These studies confirm that while access to external knowledge has a singificant impact on innovativeness (Dahlander and Frederiksen, 2012), peripheral developers still rely on core members to have their contributions approved and implement (Crowston and Scozzi, 2008). While these studies echo the findings made with the onion model, the core-periphery continuum can provide further detail about the relationship between tasks and structure by focusing on an actor's relative position in a network rather than generalising at the task cluster level. Removing the task clusters makes it possible to explore the balance between core and peripheral characteristics needed to facilitate problem solving.

Indeed, Cattani and Ferriani (2008), found that individuals located between the core and periphery (semi-peripheral actors) are often more innovative due to their proximity to both core and peripheral actors. Dahlander and Frederiksen (2012) find an

inverted U-shaped relationship between coreness and innovation, noting that semi-peripheral actors have access to external sources of information and have the social capital to mobilise resources internally. Figure 1 explains the relationship between an actor's coreness and their innovativeness. Problem-solving tasks such as bug fixing may, therefore, be best carried out by individuals located in the semi-periphery and not the periphery as the onion model suggests. We therefore argue that bug fixing activities, as with other innovative activities, will exhibit the same inverted U-shaped relationship between coreness and innateness as shown in Figure 1. This leads to our second hypothesis:

H2: *There is an inverted U-shaped relationship between an individual's 'coreness' and bug fixing contributions.*

2.3. Task and structure evolution in OSS development

The third approach looks at the evolution of OSS tasks and structure over time, focusing on changes in an individual's position within the community's structure. For instance, empirical studies have outlined a 'joining script' where individuals move from the periphery to the core through their contributions to the project (Jergensen et al., 2011; von Krogh et al., 2003). The joining script emphasises the attainment of centrality by individuals as they develop new skills and knowledge through participation (Barcellini et al., 2014). The change in roles is based on the learning process of each individual and the current understanding of legitimate peripheral participation and situated learning (Lave and Wenger, 1991). These observations, however, rely on a static view of OSS community structure, where task-related clusters are permanent while individuals move between them.

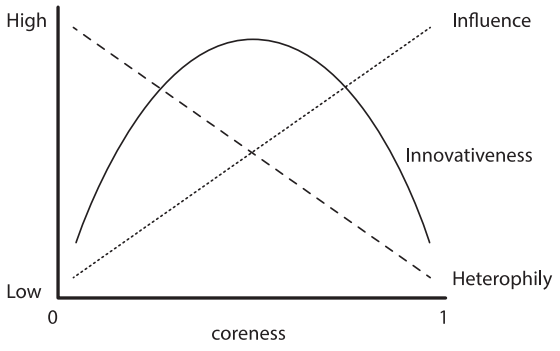

Figure 1. The relationship between coreness, heterophily, community influence and innovativeness. The closer a node is from the core (1), the higher the level of influence and the lower their access to multiple sources of information (heterophily). The further a node is from the core (0), the lower the level of influence and higher level of heterophily. Innovativeness increases as nodes achieve a balance of influence and heterophily.

It is important to take into account the stage at which an OSS project is, and the effect this may have on the tasks required. For instance, OSS projects often begin with an initiation stage where an individual is typically motivated by a 'personal itch' (Wu and Lin, 2001), leading to the creation of a working prototype (Mockus et al., 2002), and then, to its public release (Jergensen, 2001). Collaborative development will only begin once the working code is released to the public, leading to the emergence of a number of different tasks, such as general development, software testing and bug fixing (Saini and Kaur, 2014). It is after the public release of the software, and the community of users reaches critical mass, that the project moves from a centralised 'cathedral' structure to a decentralised 'bazaar' structure (Capiluppi and Michlmayr, 2007). A project's onion-like structure, and the tasks in them, may, therefore, change over time as the project progresses through each stage in its life cycle.

While each OSS project will have its own development methodology and life cycle (Tiwari, 2011; Saini and Kaur, 2014), there is a general agreement that some tasks become critical at different stages (Jergensen, 2001; Wu and Lin, 2001). When comparing OSS project life-cycles, a significant number of these studies place bug fixing activities later on in the process (Saini and Kaur, 2014). What this means is that, once the project enters a decentralised model and more people participate, these new entrants will begin to contribute as bug fixers. It is in this last step that we see that bug fixing happens at a much later time than the general commits. This therefore leads us to our third hypothesis:

H3a: *Bug fixing activities will increase over time.*

H3b: *Bug fixing contributions by the peripheral cluster will increase over time.*

It is worth noting that as we are using data from a single project, the generalisability of our results will be limited. Our findings, thus, should be interpreted within the context of LLVM to gain insights about our data. Caution is needed if they are applied to other OSS projects.[1]

3. Methods

3.1. Research design

The objective of this study is to improve our understanding of the relationship between community structure and tasks by addressing the following

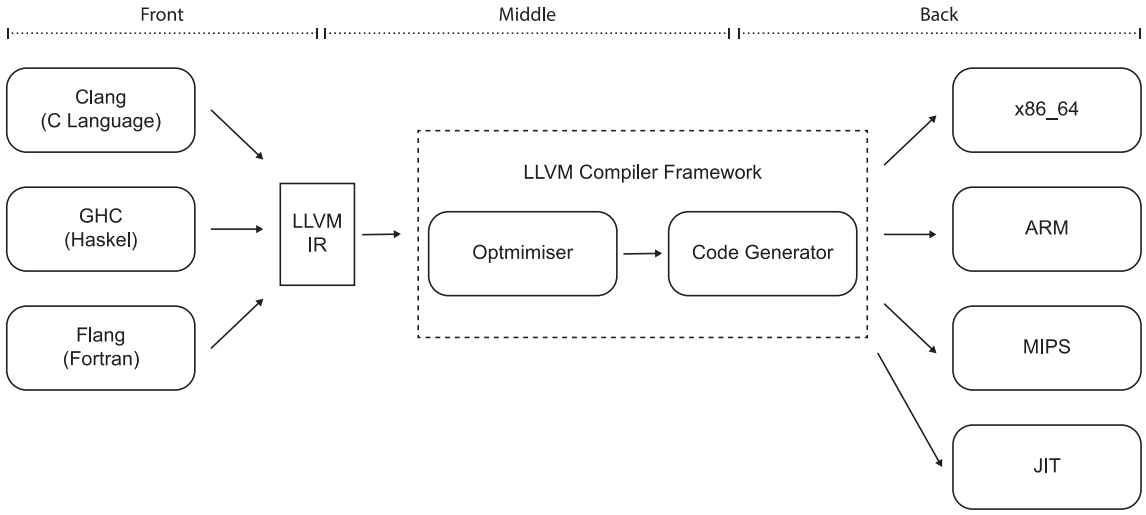

Figure 2. LLVM schema, adapted from Lattner (2008). Frontend language-specific inputs represent separate open source projects.

research questions: (1) Do peripheral developers specialise in bug fixing? (2) Is there a relationship between centrality and bug fixing and (3) Do tasks-based structures change over time?

To address these questions, we employ a Type 1 single case study approach, utilising a critical case to challenge current dominant understanding (Yin, 2018). In selecting a single case study approach, our research follows the dominant approach in OSS community studies (Crowston et al., 2012). Due to their online nature, OSS projects provide the researcher with rich contextual and historical data (von Hippel, 2005; Yin, 2018). We rely on technical contributions as the primary data source for calculating community structure and identifying bug fixing contributions. This is because, as Jergensen et al. (2011) found, there is a significantly small incidence of developers progressing from the social to the technical areas. Hence, for the purpose of this study, the boundaries of the community structure set around the repository as previously explained in Section 2.

3.2. Case selection

We selected the LLVM open source project as our critical case due to its wide adoption, the high levels of firm involvement, and the rich availability of data. As with the Linux project, the LLVM receives a large percentage of technical contributions from large firms such as Apple and IBM (Bieneman and Barton, 2019). The project was classified as one of the most popular projects on GitHub in 2018 and its repository log consists of 163,726 commits spanning a period from May 2003 to April 2018, when the data were collected.

LLVM is a compiler infrastructure project that provides a set of tools to turn high level code into machine code. It began as a Post Graduate project by Chris Lattner, along with his supervisor Vikram Adve, with the first working version being released on October 2003 under the University of Illinois at Urbana-Champaign's open source licence. While Lattner originally did not expect the project to be popular in the OSS community (LLVM, 2007a), it received significant commercial interest from the start, with representatives from Apple, Google, Adobe, HP and Cisco attending the first LLVM developers' conference in 2007 (LLVM, 2007b). Lattner has since gone on to work at Apple, focusing exclusively on integrating LLVM into their internal compilers, leading to its current use on all of Apple's operating systems.[2]

The project implements a new approach in the design of software compilers by applying the intermedia representation (IR) technique in production level compiling (Lattner, 2008). The structure of an LLVM-based compiler can be divided into three key sections, the front, middle and back end (Figure 2). The frontend generates the IR. Second, the middle part analyses and optimises the code in the IR file. Third, the backend transforms the IR into machine code (Chisnall, 2017). Since the end result relies on an IR file, this approach provides developers with the flexibility to use any frontend programming language. Given this level of flexibility, firms have implemented the LLVM tools to create their own compilers. Some of its users include SONY for their PlayStation 4 games, Adobe for their Hydra Language and Intel for the Open Computing Language (LLVM, n.d.).

3.3. Data collection and data sets

The Git software tool was used to download the repository and extract the project's log. For the purpose of this study, the metadata extracted from the project's log was limited to the date of the commit, its author's email address and its description. During the extraction of the log, data cleaning was required in order to remove double entries and maintenance activities such as code merges. This resulted in a total of 163,726 submission to the repository by a total of 998 unique developers.

From the cleaned project log, two data sets were created. The first one is a contribution data set containing a list of all 163,726 changes to the repository, along with the date, title and classification (general contribution or bug fix). The second one is a user data set containing a list of all 998 developers along with their total number of general contributions and bug fixes.

A supervised machine learning decision tree (Zhang, 2004) algorithm was used to classify each contribution as either a general contribution or a bug fix. Our approach was inspired by that of Sandusky et al. (2004), where we use regular expressions to identify bug fixes by searching for generic term like 'fix*' and 'bug*' in the commit description. The first set of results was checked manually for accuracy and an initial training set was created. A wider set of keywords was extracted from the training set and the search process was repeated once more. The process of identifying new key words, expanding the dictionary, and conducting the search was repeated several times until no new entries were identified. A sample of new search results was manually inspected after each search to ensure accuracy of results. Within the search, a decision tree was used to identify descriptions that would result in false positives, such as 'Do not fix' or 'this is a bug free implementation'. Our algorithms were written in Python and utilised NTLK (Bird et al., 2009) for language analysis.

3.4. Data analysis

Utilising the user data set, we created two conceptualisations of the core-periphery structure. First, using Bradford's Law of Scatter (Bradford, 1934; Crowston et al., 2006; Wei et al., 2017), we classify all developers into three main clusters: the periphery (here referred to as *bl1*), the semi-periphery (*bl2*) and the core (*bl3*). Second, we then calculated each developer's coreness index (*CI*) (Borgatti and Everett, 1999) by applying the Lorenz Curve calculation to the total number of contributions (general contributions and bug fixes) made to the project. The Lorenz curve plots income distribution in a given population (Gastwirth, 1971) and is used to calculate the GINI coefficient which measures inequality (Gastwirth, 1972). Both the Lorenz curve and the GINI index have been used to measure participation inequality in open source projectors and determine its core-periphery structure (Lakhani and von Hippel, 2003; Crowston and Howison, 2005; Kuk, 2006). An example of its application to calculate the coreness index is presented in Table 3.

Table 3. Example application of the Lorenz curve calculation to get the coreness index

	c	***pc***	***cpc*** (coreness)
Node 1	24	0.024	0.024
Node 2	34	0.034	0.058
Node 3	345	0.347	0.405
Node 4	236	0.237	0.642
Node 5	356	0.358	1.000

Where *c* is the number of contributions made by the developer, *pc* is the percentage out of all contributions and *cpc* is the cumulative percentage. We use the *cpc* value as the coreness index. In our example, node 1 is in the periphery and node 5 is in the core.

To test hypotheses 1 and 2, we performed an OLS regression analysis of the functional form below on the data set of 998 users:

$$fixes = \alpha + \beta_1 CI + \beta_2 CI^2 + \gamma length + \varepsilon \quad (1)$$

where *fixes* is the natural logarithm of the number of fixes, *CI* and CI^2 are the coreness index and its squared term, *length* is the natural logarithm of the number of days the user is active. We include the squared term of the coreness index to allow for its non-linear relationship with the number of fixes. For robustness check, we replace *CI* with *DC*, a dummy variable for each cluster, with $k = 2$.

To test hypothesis 3, we performed a panel data analysis of the functional form below on the contribution data set. To obtain this data set, we collate the number of contributions submitted by each user at the different time (days) of submission.

$$fixes_{i,t} = \alpha + \sum_{k=1}^{K} \beta_k DC_{k,i,t} + \gamma_2 t + \sum_{k=1}^{K} \theta_k DC_{k,i,t} t + \varepsilon \quad (2)$$

where *DC* is a dummy variable for each cluster, with $k = 2$, t is a time trend, and subscripts i and t indicate user i at time t. A random effect generalised least squares regression technique is employed.

4. Results and analysis

4.1. The periphery and bug fixes

The first set of results addresses our first research question on whether peripheral developers specialise

Table 4. Descriptive statistics of the LLMV project, with the number of active users, total contributions and fixes for each cluster

Cluster	Users	Contributions	Fixes
bl3	8	54,247	9,280 (0.206)
bl2	39	54,604	8,818 (0.193)
bl1	951	54,875	10,088 (0.225)
Total	998	163,726	28,186 (0.208)

Where *bl1* is the periphery, *bl2* is the semi-periphery and *bl3* the core.

in bug fixing activities at the cluster level. Table 4 shows the number of developers, the total number of contributions, and the number of fixes for the periphery (*bl1*), semi-periphery (*bl2*) and the core (*bl3*). As with previous studies (Crowston et al., 2006), contributions to the repository are highly centralised, with the eight developers classified as core submitting 54,248 contributions to the repository. The results show that 20.8% of all contributions were classified as being bug fixes. In addition, when looking at bug fixes in general, we see that all clusters engage in bug fixing to a similar extent, between 20% and 25% of their respective contributions.

Due to the way in which the clusters were identified we expected the level of contributions to be similar; roughly 1/3 of the total for each cluster. Based on the literature (Wei et al., 2017), however, there should be a significant difference in bug fixing contributions between the core and the periphery, as core developers may focus on general contributions to the repository while the peripheral developers are set to contribute significantly higher levels of bug fixes. While peripheral developers (cluster *bl1*) did submit a higher number of fixes both in real terms and as a percentage of contributions, all clusters submitted similar proportion of fixes, accounting for around 20% of all contributions. From this first set of results, the difference in bug-fixing patterns between each cluster is not large enough to suggest specialisation.

4.2. Coreness and bug fixing

For the second set of results reported in Table 5, we can identify whether our data support the first two hypotheses. We used non-linear regression to estimate the relationship between coreness (*CI*) and bug fixing contributions. The parameter estimated of coreness is positive and statistically significant at the 1% level, denoting a positive relationship between coreness and the number of fixes. Therefore, at this point, we can infer that for LLVM, core developers appear to contribute more to the number of fixes,

Table 5. This table reports the OLS regression results of the relationship between coreness and fixes

Variables	1	2
CI	15.2092***	
	(0.413)	
CI^2	–11.6264***	
	(0.640)	
lnlength	0.1575***	0.3415***
	(0.009)	(0.013)
bl2		2.8442***
		(0.162)
bl3		4.0670***
		(0.326)
_cons	0.1078**	–0.1898***
	(0.045)	(0.072)
N	998	998
R2	84.63%	59.71%

User data are used in this analysis. The dependent variable is the log of the number of fixes. Column 1 shows the results with CI. Column 2 shows the results with clusters. *CI* is the coreness index, CI^2 is the squared value of *CI*, *lnlength* is the log of the number of days the user is active in this project, _cons is a constant term. Dc2 and dc3 are dummies for users in cluster 2 and 3, respectively, with cluster 3 being at the core. *N* is the number of observation, R2 is the adjusted R-squared, standard errors in parentheses, *,**,***: significant at the 10%, 5% and 1% levels.

which does not support hypothesis 1. This finding reinforces our initial data analysis in Section 5.1, where there seems to be no sharp distinction between the developer groups. This is an interesting result as it shows that it is not always the case that periphery developers are more active in bug fixing compared to their core peers.

Notwithstanding, as the relationship between the two could be non-linear, we need to look at the parameter of CI^2 as well. As in hypothesis 2, we are interested in determining if there is an inverted U-shaped relationship between an individual's 'coreness' and bug fixing contributions. Our results confirm that there is an inverted U-shaped relationship between *CI* and the number of bug fixes, with the vertex of the parabola being at 0.6 (Figure 3). In the main regression analysis (reported in Table 5), we include the squared term of the *coreness* index to account for this non-linear relationship. The coreness threshold which reflects a change in this relationship is 0.654 (see Table 5, column 1). Below this threshold, the higher the index, the larger the number of fixes. Beyond this threshold, the number of fixes decreases corresponding to greater degree of centrality. At this stage, we can confirm that our data support hypothesis 2, and complement the finding related to hypothesis 1. Testing hypothesis 1 alone may not be able to paint the whole picture of the relationship, as

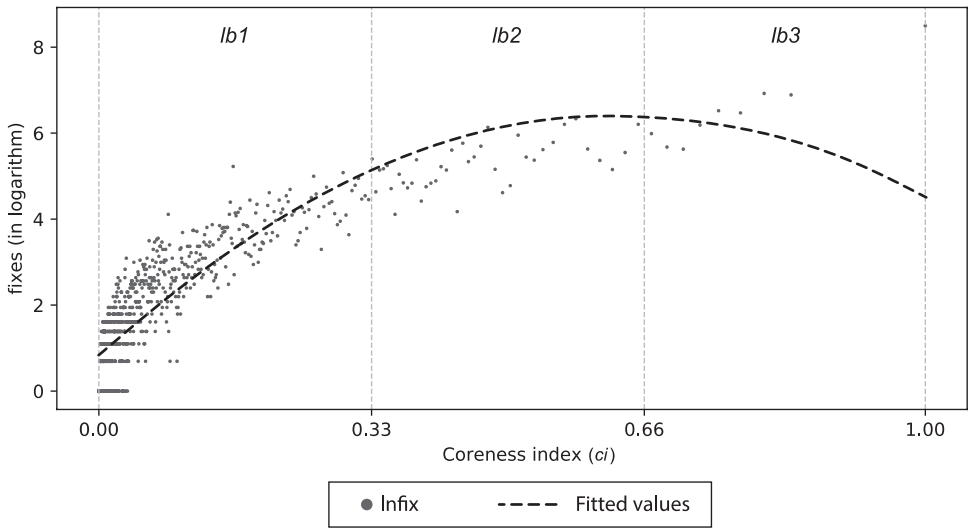

Figure 3. Fitted quadratic regression line of the log of the number of fixes on the *CI*.

at least in LLVM, the non-linear relationship shows that hypothesis 1 is partly supported.

For robustness check, we replace *CI* in model 1 with *DC*, a dummy variable for each cluster, with $k = 2$. We obtain the functional form as follows.

$$fixes = \alpha + \sum_{k=1}^{K} \beta_k DC_k + \gamma length + \varepsilon \qquad (3)$$

In column 2 of Table 5, the positive relationship between *coreness* and fixes is confirmed in this model, once again not supporting hypothesis 1. The positive parameters of the cluster dummies reveal that core developers actually contribute more to bug fixing. It is worth noting that, with the dummy variables, we are unable to test for the non-linear relationship in hypothesis 2. Hence, the results should be interpreted together with the coreness index in column 1.

The second set of results is in line with previous studies looking at the relationship between coreness and problem solving (Perry-Smith and Shalley, 2003; Cattani and Ferriani, 2008). Our findings suggest there is also an optimal level of coreness that facilitates bug fixing, located within Bradford's semi-peripheral cluster. There may be two interpretations of this results. The first is that those in the semi-periphery are in an ideal location to be able to understand the software and also have access to novel ideas in order to fix the software, similar to the cosmopolitan cluster as described by Dahlander and Frederiksen (2012). A second interpretation, however, can also be that there is an attainment of centrality, where those in the semi-periphery have been contributing longer to the project, and therefore, have gained their coreness as a result of their bug fixing activities (Barcellini et al., 2014).

4.3. Bug fixes over time

For the final set of results, we explore the relationship between the age of the project and the number of fixes being submitted to the repository to see whether tasks evolve over time. While the first set of results focused on a snapshot of the project, here, we explore changes in the number of bug fixes over time. We do this by first investigating whether there are any changes in the total number of bug fixes being submitted to the project over time. Second, we look at changes in bug-fixing patterns at the cluster level to identify whether the prominence of bug fixing changes between clusters.

At the project level, we see that the number of fixes submitted to the project have increased over time in real terms (Figure 4). This increase is in line with both an increase in general contributions and an increase in developers contributing to the project. When taking the number of bug fixes as a percentage of all contributions to the repository, however, we find an increase during the first half of the project's life cycle, and a decrease thereafter. The second half of LLVM's life cycle may have dominated the first one, as Table 6 also reports that overall, the number of fixes decreases over time. The result is statistically significant at the 1% level, although the economic impact is rather trivial. Hence, hypothesis 3a is not supported by LLVM's data. These results suggest that the project may have entered a different stage in its life cycle which may require fewer bug fixes. This change in stage may also signal a change in the prominence of bug fixing activities.

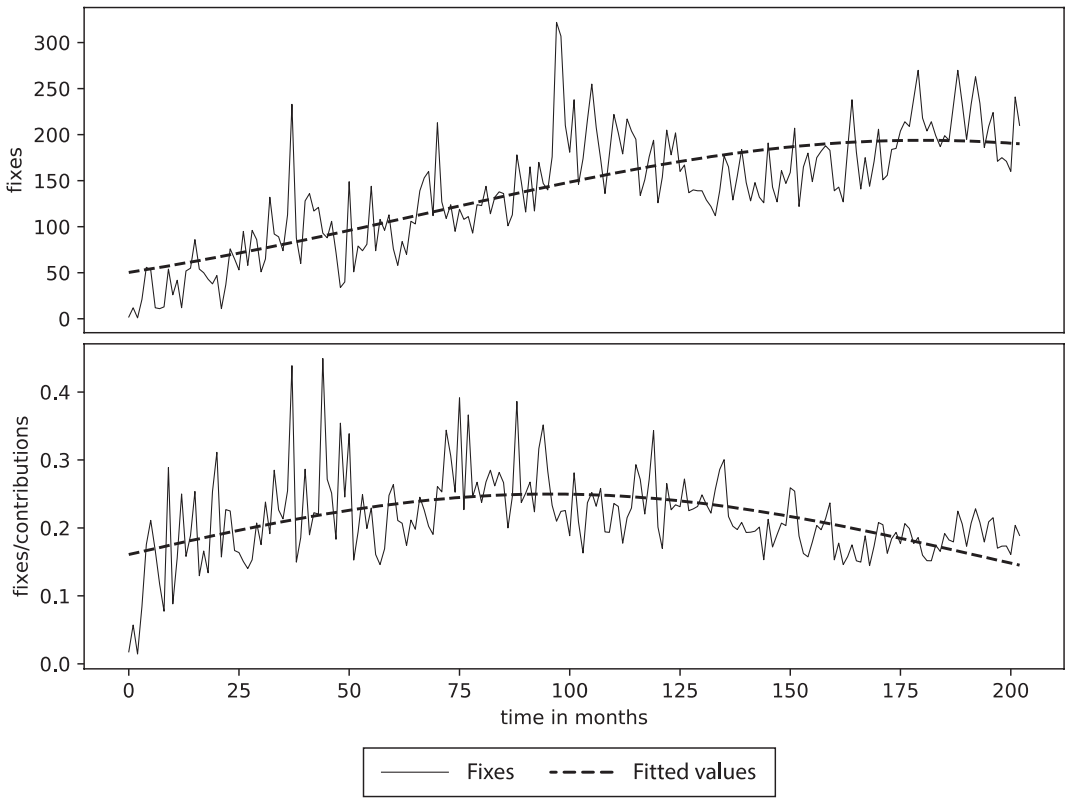

Figure 4. Total number of fixes submitted to the repository over time for each month (above) and as a percentage of total contributions (below).

Table 6. This table reports the panel regression results of the relationship between coreness and fixes over time

Variables	Parameters
t	–0.00004***
	0.000
dc2	0.46801***
	(0.123)
dc3	4.94703***
	(0.127)
dc2*t	–0.00002***
	(0.000)
dc3**t*	–0.00024***
	(0.000)
_cons	1.13001***
	(0.082)
N	59,527
R2	10.11%

Contribution data are used in this analysis. The dependent variable is the number of fixes, *t* is a time variable, dc2 and dc3 are dummies for users in cluster 2 and 3, respectively, with cluster 3 being at the core. *N* is the number of observation, R2 is the overall R-squared, standard errors in parentheses, _cons is a constant term, *,**,***: significant at the 10%, 5% and 1% levels.

In Table 6, we can observe whether there are changes to the number of fixes being submitted to the project at the cluster level. Our analysis shows this to be the case. In particular, relative to *bl1* (the periphery), we observe a larger decrease in fixes over time for users in *bl3* (core). A similar finding is reported for users in *bl2* (semi-periphery). The results are statistically significant, however, with a small magnitude, which is to be expected due to the data being measured at daily frequency. To this end, in LLVM, hypothesis 3b is also not supported by the data.

To a certain extent, this last set of results supports findings from studies on the duality of the core and periphery, where bug fixing activity moves to the peripheral cluster as the project progresses, (Capiluppi and Michlmayr, 2007). This set of results therefore implies that specialisation of bug fixing activities by the peripheral cluster emerges over time as the project moves through its life cycle.

5. Discussion and conclusion

In this study, we investigate the link between tasks and community structure by looking at bug-fixing patterns in the LLVM OSS project. The extant literature on OSS developer communities has stated that

there is a long-standing relationship between community structure and the roles that individuals carry out within the project. This has been exemplified by the core-periphery and onion model that give specific roles or functions depending on an individual's position within the structure. Despite the ubiquity of these models, there is little empirical evidence to support many of the inferences being made. To address this, we used a most common form of development task, bug fixing, to identify its relationship to community structure.

This study makes three significant findings. First, it finds that bug fixing activities are carried out by most individuals in the OSS community irrespective of their structural position. Second, we find, however, that there is an inverse U-shaped relationship between 'coreness' and bug fixes, where the number of fixes submitted increases with centrality but then tapers off as it gets closer to the core. Finally, we find evidence to suggest that bug-fixing patterns evolve over time, with those in the core and semi-periphery carrying out fewer bug fixes over time when compared to those in the periphery. These results provide a deeper understanding of the relationship between tasks and community structure.

Our first research question focuses on whether peripheral developers specialise in bug fixing as suggested in the onion model. The onion model has emerged as an important tool for understanding how OSS communities work, with each layer visually representing task groups, their size and their impact on the development process. Linking bug fixing activities to peripheral developers has been one of the consistent characteristics of the onion model. Our study, however, finds no empirical evidence to support this assumption. When looking at all contributions made to the repository, irrespective of time, we find that both core and peripheral subgroups contribute to bug fixing activities in roughly equal measures. Our findings do not support the link between bug-fixing tasks and structure as presented in the onion model.

Given that the intuitive model offered no clear evidence of tasks specialisation, we then utilised the continuous model to address our second research question on the relationship between centrality and bug fixing activities. Our study finds that the continuous model proposed by Borgatti and Everett (1999) provides a more accurate link between an individual's position within the community and their ability to carry out certain tasks. Our study provides empirical evidence that there is an optimal level of 'coreness' that leads to a higher level of bug fixing contributions. In doing so, our study supports previous findings in social network theory linking network structure to problem-solving ability (Dahlander and Frederiksen, 2012). As a task-related structure, therefore, our study provides empirical evidence to support the continuous model linking 'coreness' with bug fixing.

Finally, both the intuitive and continuous core-periphery models used are a static representation of structure. For our third research question, we assess if these structures change over time. Our study provides empirical evidence suggesting that project structures and tasks may evolve over time. Studies have long acknowledged the different stages in OSS development processes (Capiluppi and Michlmayr, 2007), but these have not been taken into account when looking at their implication of task-related structures. Our findings show that, while bug fixing activities decreased over time, those carried out by peripheral developers decreased at a lower rate than those closer to the core. Our findings add to the dynamic view of task-related structures by opening up the idea that the prominence of tasks evolves over time. When relating this to the onion model, therefore, it could be that the model symbolises the community structure at a specific point in time towards the later stages of the project's life cycle. A different onion model could potentially emerge when looking at earlier stages, such as the initial prototype stage before the software is released to the general public.

5.1. Managerial implications

The findings from this study have implications for both project owner and core developers. First, the results in this study provide project owners with insights on how to manage participation in specific tasks within the development process. A significant portion of the literature on open source membership management focuses on the process of gaining access and increasing authority within a project as contributors progress through the onion model (Dahlander and O'Mahony, 2011; Jergensen et al., 2011). Our findings provide an additional trajectory which emphasises access to knowledge and influence required to carry out specific development tasks. To this point project leaders should recognise the importance of facilitating the transition from periphery to semi-periphery. This could be done by curating and providing simpler tasks for newcomers in order to help them gain the required knowledge and skills to contribute with more critical submission in the future.

A second managerial implication involves the change in roles for project leaders. Past studies have focused on how an individual's role in the project changes as they gain centrality (von Krogh et al., 2003; Dahlander and O'Mahony, 2011). The

findings in this study suggest that, while an individual's position within the community structure may remain the same, their role in the project may change over time. This was particularly evident in the observed decrease in bug fixing contributions by the core, implying that core developer and project leaders may shift from a technical to managerial roles over time as the number of contributors increases. Project leaders and core developers should, therefore, be aware of the need to acquire non-technical skills to manage the project through its life cycle.

5.2. Limitations

This study is not without its limitations. The two main limitations are its level of analysis and its focus on community structure. Our study focuses on bug fixing activities within a singular community (software developers) active within one production area (the repository). Our specific focus does not take into account other tasks included in the onion structure, such as community manager or passive user. This therefore presents a limitation in that the core-periphery structure is measured within narrow boundaries. Extending these boundaries to include project level tasks may provide different results. The second limitation is that it does not take into account the socio-technical structure of complex open source projects (Amrit, 2008). This is to say, some of the projects are highly modular and developers often limit their contributions to specific modules. Future studies could, therefore, aim to observe similar patterns in both the core section of the code and how these differ from the peripheral modules. Taking the socio-technical structure into account, specifically at modularity, may provide further insights into task-related structures.

References

Amrit, C. (2008) *Improving Coordination in Software Development through Social and Technical Network Analysis*. Enschede: University of Twente. https://research.utwente.nl/en/publications/improving-coordination-in-software-development-through-social-and

Bach, P. and Twidale, M. (2010) Social participation in open source: what it means for designers. *Interactions*, **17**, 3, 70–74. Available at: http://dl.acm.org/citation.cfm?id=1744177.

Barcellini, F., Détienne, F., and Burkhardt, J.M. (2014) A situated approach of roles and participation in open source software communities. *Human-Computer Interaction*, **29**, 3, 205–255.

Becker, M.H. (1970) Sociometric location and innovativeness: reformulation and extension of the diffusion model. *American Sociological Review*, **35**, 2, 267–282. https://doi.org/10.2307/2093205.

Bieneman, C. and Barton, K. (2019) *How to Contribute to LLVM*. Available at: https://youtu.be/C5Y977rLqpw (accessed 30 March 2020).

Bird, S., Klein, E., and Loper, E. (2009). *Natural Language Processing with Python: Analysing Text with the Natural Language Toolkit*. Sebastopol, CA: O'Reilly Media.

Borgatti, S.P. and Everett, M.G. (1999) Models of core/periphery structures. *Social Networks*, **21**, 375–395.

Bradford, S.C. (1934) Sources of information on specific sources. *Engineering: An Illustrated Weekly Journal*, **137**, 3550, 176–180.

Burt, R.S. (2004) Structural holes and good ideas. *American Journal of Sociology*, **110**, 2, 349–399. https://doi.org/10.1086/421787.

Capiluppi, A. and Michlmayr, M. (2007) From the cathedral to the bazaar: an empirical study of the lifecycle of volunteer community projects. *IFIP International Federation for Information Processing*, **234**, 31–44. https://doi.org/10.1007/978-0-387-72486-7_3.

Cattani, G. and Ferriani, S. (2008). A core/periphery perspective on individual creative performance: social networks and cinematic achievements in the hollywood film industry. *Organization Science*, **19**, 6, 824–844. https://doi.org/10.1287/orsc.1070.0350.

Chisnall, D. (2017) Modern Intermediate Representations (IR). In: *LLVM Summer School*, Paris.

Christian, J. (2016) *The User Organisation: Governance and Structure in an Open Source Project*. University of Brighton. Available at: http://eprints.brighton.ac.uk/15449/.

Chu, K.M. and Chan, H.C. (2009) Community based innovation: its antecedents and its impact on innovation success. *Internet Research*, **19**, 5, 496–516.

Cox, A. (1998) Cathedrals, Bazaars and the Town Council. *Slashdot.org*, pp. 12–14. Available at: https://news.slashdot.org/story/98/10/13/1423253/featurecathedrals-bazaars-and-the-town-council.

Crowston, K. and Howison, J. (2005) The social structure of free and open source software development. *First Monday*. http://dx.doi.org/10.5210/fm.v10i2.1207

Crowston, K. and Howison, J. (2006) Hierarchy and centralization in free and open source software team communications. *Knowledge, Technology & Policy*, **18**, 4, 65–85. https://doi.org/10.1007/s12130-006-1004-8.

Crowston, K. and Scozzi, B. (2008) Bug fixing practices within free/libre open source software development teams. *Journal of Database Management*, **19**, 2, 1–30. https://doi.org/10.4018/jdm.2008040101.

Crowston, K., Wei, K., Li, Q., and Howison, J. (2006) Core and periphery in Free/Libre and open source software team communications. In: *Proceedings of the 39th Hawaii International Conference on System Sciences*, Kauia, HI. pp. 65–85.

Crowston, K., Wei, K., Howison, J., and Wiggins, A. (2012) Free/Libre open-source software development: what we know and what we do not know. *ACM Computing Surveys*. **44**, 2, 1–35. https://doi.org/10.1145/2089125.2089127.

Dahlander, L. and Frederiksen, L. (2012) The core and cosmopolitans: a relational view of innovation in user communities. *Organization Science*, **23**, 4, 988–1007. https://doi.org/10.1287/orsc.1110.0673.

Dahlander, L. and O'Mahony, S. (2011) Progressing to the center: coordinating project work. *Organization Science*, **22**, 4, 961–979.

Dahlander, L., Frederiksen, L., and Rullani, F. (2008) Online communities and open innovation: governance and symbolic value creation. *Industry and Innovation*, **15**, 2, 115–123. https://doi.org/10.1080/13662710801970076.

Daniel, S. and Stewart, K. (2016) Open source project success: resource access, flow, and integration. *Journal of Strategic Information Systems*, **25**(3), 159–176. https://doi.org/10.1016/j.jsis.2016.02.006.

Fielding, R.T. (1999) Shared leadership in the Apache Project. *Communications of the ACM*, **42**, 4, 42–43. https://doi.org/10.4135/9781483349107.n9.

Franck, E. and Jungwirth, C. (2003) Reconciling rent-seekers and donators - the governance structure of open source. *Journal of Management and Governance*, **7**, 4, 401–421. https://doi.org/10.1023/A:1026261005092.

Gastwirth, J.L. (1971) A general definition of the Lorenz Curve. *Econometrica*, **39**, 6, 1037–1039. https://doi.org/10.1090/ulect/038/03.

Gastwirth, J.L. (1972) The estimation of the Lorenz Curve and Gini Index. *The Review of Economics and Statistics*, **54**, 3, 306. https://doi.org/10.2307/1937992.

Glass, R. (2003) A sociopolitical look at Open Source. *Communications of the ACM*, **46**, 11, 21–23. https://doi.org/10.1109/MC.2004.38.

Granovetter, M.S. (1973) The strength of weak ties. *American Journal of Sociology*, **78**, 6, 1360–1380. https://doi.org/10.4324/9780429499821-43.

Hargadon, A.B. (2006) Bridging old world and building new ones: towards a microsociology of creativity. In: *Creativity and Innovation in Organizational Teams*. Brighton, UK: Psychology Press. pp. 199–216.

Hienerth, C., Von Hippel, E., and Berg Jensen, M. (2014) User community vs. producer innovation development efficiency: a first empirical study. *Research Policy*, **43**, 1, 190–201. https://doi.org/10.1016/j.respol.2013.07.010.

Hossain, L. and Zhu, D. (2009) Social networks and coordination performance of distributed software development teams. *The Journal of High Technology Management Research*, **20**, 1, 52–61.

Jensen, C. and Scacchi, W. (2007) Role migration and advancement processes in OSSD projects. In: *29th IEEE/ACM International Conference on Software Engineering (ICSE'07)*, Minneapolis, MN. pp. 364–374.

Jergensen, C., Sarma, A., and Wagstrom, P. (2011) The onion patch: migration in open source ecosystems. In: *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering - SIGSOFT/FSE '11*, p. 70. https://doi.org/10.1145/2025113.2025127.

Jergensen, N. (2001) Putting it all in the trunk: incremental software devevolpment in the FreeBSD open source project. *Information Systems Journal*, **11**, 321–336.

Kilamo, T., Hammouda, I., Mikkonen, T., and Aaltonen, T. (2012) From proprietary to open source – growing an open source ecosystem. *Journal of Systems and Software*, **85**, 7, 1467–1478. https://doi.org/10.1016/j.jss.2011.06.071.

Kuk, G. (2006) Strategic interaction and knowledge sharing in the KDE developer mailing list. *Management Science*, **52**, 7, 1031–1042. https://doi.org/10.1287/mnsc.1060.0551.

Lakhani, K.R. (2006) *The Core and the Periphery in Distributed and Self-Organizing Innovation Systems*. Cambridge, MA: Massachusetts Institute of Technology. https://dspace.mit.edu/handle/1721.1/34144

de Laat, P.B. (2007) Governance of open source software: state of the art. *Journal of Management and Governance*, **11**(2), 165–177. https://doi.org/10.1007/s10997-007-9022-9.

Lakhani, K.R. and von Hippel, E. (2003) How open source software works: user-to-user assistance. *Research Policy*, **32**, 6, 923–943.

Lattner, C. (2008) LLVM and Clang: Next generation compiler technology LLVM: low level virtual machine. *The BSD Conference*, Ottawa, Canada.

Lave, J. and Wenger, E. (1991) *Situated Learning: Legitimate Peripheral Participation*. Cambridge: Cambridge University Press.

Lee, G.K. and Cole, R.E. (2003) From a firm-based to a community-based model of knowledge creation: the case of the Linux Kernel development. *Organization Science*, **14**, 6, 633–649.

Lehmann, F. (2004) FLOSS developers as a social formation. *First Monday*. http://dx.doi.org/10.5210/fm.v9i11.1186

Lin, Y. (2005) The future of sociology of FLOSS. *First Monday*, **2**, 1–5.

LLVM. (2007a) *2007 LLVM Developers' Meeting: V. Adve & C. Lattner, "A brief history of LLVM"*. Available at: https://youtu.be/FNtmemyeEHY.

LLVM. (2007b) *LLVM Developers' Meeting Proceedings*. Available at: https://llvm.org/devmtg/2007-05/ (accessed 30 March 2020).

LLVM. (n.d.) *LLVM Users*. Available at: http://llvm.org/Users.html (accessed 31 March 2020).

Madey, G., Freeh, V., and Tynan, R. (2004) Modeling the Free/Open source software community: a quantitative investigation. *FreeOpen Source Software Development*, 203–220. Available at: http://www.amazon.com/Free-Open-Source-Software-Development/dp/1591403693.

McPherson, M., Smith-Lovin, L., and Cook, J.M. (2001) Birds of a feather: homophily in social networks. *Annual Review of Sociology*, **21**, 7, 415–444.

Mockus, A., Fielding, R., and Herbsleb, J. (2002) Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, **11**, 3, 309–346. https://doi.org/10.1145/567793.567795.

Monteiro, E., Østerlie, T., Rolland, K.H., and Røyrvik, E. (2004) *Keeping It Going: The Everyday Practices of Open Source Software*. Working paper. Trondheim, Norway: Department of Computer and Information

Science, Norwegian University of Science and Technology, pp. 1–32.

Nakakoji, K., Yamamoto, Y., Nishinaka, Y., and Ye, Y. (2002) Evolution patterns of open-source software systems and communities. In: *Proceedings of the international workshop on Principles of software evolution - IWPSE '02* (January), 76–85.

O'Mahony, S. and Ferraro, F. (2007) The emergence of a governance structure in an open source community. *Academy of Management Journal*, **50**, 5, 1079–1106.

Perry-Smith, J. and Shalley, C. (2003) The social side of creativity: a static and dynamic social perspective. *Academy of Management Review*, **28**, 1, 89–106.

Raymond, E.S. (1998) The cathedral and the bazaar. *First Monday*, **3**(2), 1–45.

Rullani, F. and Haefliger, S. (2013) The periphery on stage: the intra-organizational dynamics in online communities of creation. *Research Policy*, **42**, 4, 941–953. https://doi.org/10.1016/j.respol.2012.10.008.

Sack, W., Détienne, F., Ducheneaut, N., Burkhardt, J.-M., Mahendran, D., and Barcellini, F. (2006) A methodological framework for socio-cognitive analyses of collaborative design of open source software. *Computer Supported Cooperative Work (CSCW)*, **15**, 2–3, 229–250.

Saini, M. and Kaur, K. (2014) A review of open source software development life cycle models. *International Journal of Software Engineering and its Applications*, **8**, 3, 417–434. https://doi.org/10.14257/ijseia.2014.8.3.38.

Sandusky, R.J., Gasser, L., and Ripoche, G. (2004) Bug report networks: varieties, strategies, and impacts in a F/OSS development community. In: *Paper Presented at the Proceedings of the ICSE Workshop on Mining Software Repositories*, Edinburgh, Scotland. pp. 80–84. https://doi.org/10.1049/ic:20040481.

Scacchi, W. (2004) Free and open source development practices in the game community. *IEEE Software*, **21**, 1, 59–66.

Schilling, M.A. (2005) A 'small world' network model of cognitive Insight. *Creativity Research Journal*, **17**, 2–3, 131–154. https://doi.org/10.1080/10400419.2005.9651475.

Singh, P.V., Fan, M., and Tan, Y. (2007) An empirical investigation of code contribution, communication participation, and release strategy in open source software development: a conditional hazard model approach. FLOSS Working Paper. Available from: http://flosspapers.org/327(i).

Tiwari, V. (2011) Software engineering issues in development models of open source software. *International Journal of Computer Science and Technology*, **2**, 2, 38–44.

Uzzi, B. (1997) Social structure and competition in interfirm networks: the paradox of embeddedness. *Administrative Science Quarterly*, **42**, 1, 35–67.

von Hippel, E. (2005) *Democratizing Innovation*. Cambridge, MA: MIT Press.

von Krogh, G., Spaeth, S., and Lakhani, K.R. (2003) Community, joining, and specialization in open source software innovation: a case study. *Research Policy*, **32**, 7, 1217–1241.

Wasserman, S. and Faust, K. (1999) *Social Network Analysis: Methods and Applications*. Cambridge, UK.: Cambridge University Press.

Wei, K., Crowston, K., Eseryel, U.Y., and Heckman, R. (2017) Roles and politeness behavior in community-based free/libre open source software development. *Information and Management*, **54**, 5, 573–582. https://doi.org/10.1016/j.im.2016.11.006.

Wu, M. and Lin, Y. (2001) Open source software development: an overview. *Computing Practices*, **34**, 6, 33–38.

Yamauchi, Y., Yokozawa, M., Shinohara, T., and Ishida, T. (2000) Collaboration with Lean Media: how open source software succeeds. In: *Proceedings of the 2000 ACM conference on Computer supported cooperative work - CSCW'00*. pp. 329–338. https://doi.org/10.1145/358916.359004.

Ye, Y. and Kishida, K. (2003) Toward an understanding of the motivation of open source software developers. In: *Proceedings of the 25th International Conference on Software Engineering*, Portland, OR. pp. 1–11.

Yin, R.K. (2018) *Case Study Researcha and Applications: Design and Methods*, 6th edn. London, UK: SAGE Publications. https://doi.org/10.1017/CBO9781107415324.004.

Zhang, H. (2004) The optimality of Naive Bayes. In: *Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference FLAIRS*, 1, 2. pp. 1–6. https://doi.org/10.1016/j.patrec.2005.12.001.

Notes

[1]We thank an anonymous referee for this suggestion.

[2]We thank an anonymous referee for their suggestion of firms involvement.

Jose Christian is a lecturer in Innovation and Entrepreneurship at CENTRIM, University of Brighton. His research interests inlcude collaborative open user innovation, online innovation communities and open source goverance.

Anh N. Vu is a lecturer in Finance at the University of Sussex Business School. Her research interests include efficiency and risk management in crowdfunding platforms and cryptocurrencies.

