Characterizing Key Developers: A Case Study with Apache Ant

Gustavo A. Oliva[1], Francisco W. Santana[2], Kleverton C.M. de Oliveira[2], Cleidson R.B. de Souza[2,3], Marco A. Gerosa[1]

[1] Department of Computer Science, University of São Paulo (USP), Brazil
{goliva,gerosa}@ime.usp.br
[2] Computing Department, Federal University of Pará (UFPA), Brazil
[3] Vale Technological Institute – Sustainable Development (ITV – DS), Brazil
{wertherjr,kleverton.macedo}@gmail.com,
cleidson.desouza@acm.org

Abstract. The software architecture of a software system and the coordination efforts necessary to create such system are intrinsically related. Making changes to components that a large number of other components rely on, the technical core, is usually difficult due to the complexity of the coordination of all involved developers. However, a distinct group of developers effectively help evolving the technical core of software projects. This group of developers is called key developers. In this paper we describe a case study involving the Apache Ant project aimed to identify and characterize key developers in terms of their volume of contribution and social participation. Our results indicated that only 25% of the developers may be considered as key developers. Results also showed that key developers are often active in the developers' mailing list and often fulfilled the coordination requirements that emerged from their development tasks. Finally, we observed that the set of key developers was indistinguishable from the set of top contributors. We expect that this characterization enables further exploration over contribution patterns and the establishment of profiles of FLOSS key developers.

Keywords: software architecture, collaboration, socio-technical analysis, mining software repositories, case study.

1 Introduction

In the 60s', Conway [6] suggested that the relationship between the architecture of a software system and the structure of the organization developing this software is homomorphic – the Conway's Law. Similarly, Parnas [17] suggested an approach, the information hiding principle, to structure the software architecture in such a way to reduce coordination needs among developers. Recently, these theoretical proposals have been corroborated by several qualitative [25, 10, 24] and quantitative [5, 4] studies.

These results basically suggest that the structure of a software system influences and is influenced by the communication and coordination efforts of the developers developing such system. Furthermore, the coordination necessary to evolve highly interconnected software components is usually greater than the effort required to evolve independent components. This seems to be the case even when well-defined APIs are used [24]. In fact, despite the rhetoric about openness, access to the technical core of a software project (the set of the most important software components on which other components rely on) is limited [22]. Apart from that, we cannot say much more about the group of developers that help evolve the technical core. Are these *key developers* the ones that communicate more to other developers in the mailing-list? Are they the ones in the core of the coordination requirements network? Are they the ones that have higher socio-technical congruence [5] when considering the mailing-list network (social activities that actually occurred) and the coordination requirements network (social activities that should have taken place)? Are they also the top committers? In a long term perspective, a better characterization of key developers should help researchers understand the process a developer undergoes in order to become a key developer.

In this paper, we describe a case study conducted with the open source project Apache Ant[1] in order to investigate the characteristics of its *key developers*, i.e., the set of developers that work on the technical core of this project. Firstly, we designed and applied an appropriate method to evaluate how limited the number of key developers is. Afterwards, we investigated whether these developers (i) were central in the mailing-list network, (ii) were central in the coordination requirements network, (iii) had a higher congruence when considering these two social networks [5], or (iv) were just the top committers. Our results indicated that only 25% of the developers were classified as key developers. Results also showed that key developers were active in the developers' mailing list and often fulfilled the coordination requirements that emerged from their development tasks (high socio-technical congruence). Finally, we observed that the set of key developers was indistinguishable from the set of top contributors.

The rest of this paper is organized as follows. In Section 2, we present our research questions. In Section 3, we present related work. Section 4 then describes the research method, including the supporting tools we used. Our results are presented in Section 5. After that, Section 6 presents a discussion of our results and describes the threats to the validity of this study. Finally, in Section 7, we state our conclusions and plans for future work.

2 Characterizing Key Developers

The relationship between the architecture of a software system and the coordination required to evolve such a system is long recognized by researchers and practitioners. For instance, the performance of software developers is related to how well they align

[1] `http://ant.apache.org`

their coordination efforts with the existing technical dependencies in the software architecture, both at the team level [25] and at the individual level [4]. Indeed, misalignment between these aspects is seen as a possible explanation for breakdowns in software development projects [2]. In other words, the relationship between software architecture and coordination suggests that the coordination effort necessary to develop highly interconnected software components is usually higher than to develop independent components. This is true even when well-defined interfaces are used among software components [24].

In any software system, there are components that are regarded as more important than others. Such components constitute the technical core of a project, i.e. the set of the most important software components on which lots of other components rely on. In this paper, we call *key developers* the set of developers who help evolve the technical core of a software system. Given the existing relationship between software architecture and coordination, we expect the access to the technical core of a software project to be limited. This aspect has already been observed in previous studies of open source projects [22]. In other words, we expected a limited number of key developers. The reason is twofold: (i) the technical core is naturally important (if someone "breaks" a core component, then several other components are likely to be affected) and (ii) the complexity of the coordination necessary to make changes to the core is high. This leads to our first research question:

RQ 1: How limited is the number of key developers in a software project?

Social interaction within software development is acknowledged as an important aspect in software projects and thus has been the subject of a series of studies [6, 17, 10, 23]. Different social processes (e.g., development of a shared understanding of the system architecture, conflict resolution, and leadership establishment) are necessary for successful projects. These social processes often involve key developers differently from the rest [8], we believe that a better characterization of such developers would be beneficial to researchers interested in collaborative software development. The investigation of key developers seems especially suitable in the context of free/libre open source software development (FLOSS development) and global software development (GSD), where social interaction data is usually available in software repositories and in the project's website. This leads us to our second research question.

RQ 2: How distinct is the participation of key developers in terms of communication and coordination?

While conducting two case studies involving the Apache Server and the Mozilla web browser respectively, Mockus *et al.* [13] proposed the following hypothesis: "*open source developments will have a core of developers who control the code base, and will create approximately 80% or more of the new functionality. If this core group uses only informal, ad hoc means of coordinating their work, it will be no larger than 10-15 people.*" As our goal in this study involves characterizing key developers, we also intend to verify whether a relaxed version of such hypothesis also holds for the Apache Ant project. More specifically, instead of looking for added functionality, we will just analyze the number of modifications made by each developer. We operationalize that by identifying the group of *top contributors*, i.e. the set of developers who performed the highest number of modifications (commits) to the project. We thus state our last research question as follows.

RQ 3: What is the contribution volume of key developers?

3 Related Work

A number of previous studies have investigated the participation of open source developers regarding their "status" position within the community. For instance, Crowston *et al.* [8] examined how the group of core developers can be empirically distinguished. The authors investigated three specific approaches, namely (i) the named list of developers, (ii) the most frequent contributors, and (iii) a social network analysis of the developers' interaction pattern. By applying these three approaches to the interactions around bug fixing for 116 SourceForge projects, the authors concluded that each approach identify different individuals as core developers. However, as in our paper, the results suggest that the group of core developers in FLOSS projects corresponds to only a small fraction of the total number of contributors. In another example, Terceiro *et al.* [26] investigated the relationship between code structural complexity and the participation level of developers (dichotomized as core and peripheral). By relying on previous studies of Robles *et al.* [18, 19], the authors split the entire studied period in 20 periods of equal duration, and for each period, they considered the 20% top committers to be the core team. They found out that core developers make changes to the source code without introducing as much structural complexity as the peripheral developers. Moreover, core developers also remove more structural complexity than peripheral developers.

Other studies have focused on investigating the characteristics and behavior of software developers from a social network analysis (SNA) perspective. De Souza *et al.* [22] investigated the ways in which development processes are somehow inscribed into software artifacts. The authors hypothesized that when developers shift from the periphery to the core of the code authorship social network, a distinct phenomenon occurs. Developers initially contribute code that performs some functionality by calling others' code and, as these developers become more important, their code start to be called by other developers. De Souza and colleagues showed a periphery to core shift within the MegaMek project, and a core to periphery shift (opposite effect) within the Apache Ant project. In another study, Oezbek *et al.* [16] investigated the patterns of interaction among the core and peripheral sets of developers in order to check the validity of the "onion model" [14]. After building social networks based on mailing lists data from 11 FLOSS projects of different domains, the authors observed that the core holds a disproportionally large share of communication with the periphery. They also state that members of the core not only show a particular intense participation, but also appears to have a qualitatively different role as well. However, such hypothesis remains to be investigated. The authors also conclude that the transition of individual mailing-list participants towards ever higher participation is qualitatively discontinuous.

4 Research Method

In order to answer the research questions defined previously, we decided to adopt a case study as our research method. A case study is a well-established empirical method aimed at investigating contemporary phenomena in their natural context [28]. More specifically, we conducted a *descriptive case study with retrospective data collection* [20]. In this case study we sought to portray the characteristics of key developers by leveraging the project's available stored data. In contrast to embedded case studies, where multiple units of analysis are studied within a case, our case study is essentially holistic, i.e. the case is studied "as a whole." In a nutshell, we focused on a particular open source project and gathered different types of information from it.

In the next subsections, we present the case study design and planning. We present the rationale for choosing the case, the supporting tools we used, and the main steps we followed.

4.1 The Case

For the case study, we needed a software project that satisfied the following requirements: (i) a software project hosted on a Subversion (SVN) repository with anonymous read access; (ii) availability of information about the development activities (change logs and communication records) during a release interval, and (iii) a number of active developers greater than 15. The first requirement exists due to constraints on the tools at our disposal. The second requirement was raised because we need development information to generate the social networks and compute volume contribution. Furthermore, we will focus our analysis on a specific release interval so as to minimize influencing factors. Finally, the third requirement came up because we need sufficient social data to answer our research questions. Hence, we decided to focus on non-small development teams: Levine and Moreland [11] defined small teams as groups of 5 to 15 individuals.

After inspecting a series of open source projects, we decided to analyze Apache Ant: it is hosted on Subversion, information about development activities is available, and 16 developers contributed to it during the studied release interval. More precisely, we investigated Apache Ant Core, which is the main Ant module. We considered a development period that ranges from release 1.6 (December 19th, 2003) until release 1.7 (December 13th, 2006). In such period, a total of 2053 commits were made by a group of 16 active developers. Apache Ant is hosted by the Apache Software Foundation and is one of the most popular open source tools for automating software build processes.

4.2 Supporting Tools

Empirical studies that mine software repositories usually require extensive tool support due to the large amount and complexity of the data to be collected, processed, and analyzed [22]. Given the different data sources required in this study, we employed a variety of tools: XFlow [21], JDX, Jung[2], and OSSNetwork [1].

XFlow. XFlow is an extensible open source tool we developed whose main goal is to support empirical software evolution analyses by considering both social and technical aspects. By bringing together these two views, the tool aims to support exploratory and descriptive case studies that call for a deeper understanding of software evolution aspects. In this study, we employed XFlow to calculate the coordination requirements network [5].

JDX. Java Dependency eXtractor is a Java library we developed to extract dependencies and compute the call-graph from Java code. The library relies on the robust Java Development Tools Core (JDT Core)[3] library, which is the Eclipse IDE incremental compiler. As a desirable consequence, JDX is able to handle Java source code in its plain form. This facilitates studies that involve processing large amounts of code mined from version control systems.

OSSNetwork. OSSNetwork is a tool we developed that (i) retrieves data from software repositories (forums, mailing lists, issue tracking systems, and chats) by parsing HTML information and (ii) generates different social networks, thus supporting the analysis of social aspects of software development. We used OSSNetwork to compute the communication network from the developers' mailing list.

Jung. Java Universal Network/Graph Framework is a Java library that provides a common and extendible language for modeling, analyzing, and visualizing data that can be represented as a graph or network. We used Jung to compute network properties, such as the eigenvector centrality of nodes (as will be detailed in the following subsection).

4.3 Main Steps

In order to answer our research questions we mined Apache Ant's development repositories, namely the version control system (Subversion) and the developers' mailing list. This whole process was divided into three main steps:

I) Identifying Key Developers. The varying complexity of software system modules requires an equally varied amount of knowledge from developers in order to complete their tasks. As we are interested in characterizing key developers, our first step was to discover which developers actually worked on the core files of the Apache Ant project. In other words, this investigation requires finding both the core of the technical network and the particular developers that worked on such core. Hence, for each Subversion revision embedded in the studied development period, we did the following sub-steps:

[2] `http://jung.sourceforge.net/`
[3] `http://www.eclipse.org/jdt/core/`

a) Generate the project's technical network. We calculated the project's static call-graph using JDX. According to Wikipedia, a static call-graph is a directed graph that represents calling relationships between subroutines in a computer program. In our context, each node represents a method and each edge (*f*, *g*) indicates that a method *f* calls a method *g* (including constructor invocations). After obtaining the call-graph, we clustered the method nodes belonging to the same compilation unit. We thus obtained a new graph in which the nodes represent the compilation units and the edges represent their calling relationship. We considered such graph to be a suitable representation of the project's technical network.

b) Finding the core of the technical network. We used the eigenvector centrality measure to find the core of the network produced in the prior step. Such measure embodies the notion that a node's importance in a network is increased by having connections to other vertices that are themselves important [15]. Indeed, we believe that a compilation unit becomes important by having connections to other compilation units that are themselves important. We calculated the centrality of each node of the network and then we performed a quartile analysis to identify the network's core. The nodes that had a centrality score equal to or larger than the third quartile (Q3) were deemed as core.

c) Computing commit coreness. In order to differentiate developers' contributions, we conceived a measure for computing the *commit coreness*. This measure is calculated based on the number of modified core artifacts, thus enabling us check whether a developer actually contributed to the technical core or just made peripheral changes:

$$Coreness(commit) = \frac{Number\ of\ core\ files\ in\ the\ commit}{Total\ number\ of\ files\ in\ the\ commit}$$

When *commit coreness* was greater than or equal to 0.5, we considered it to be a core commit. In fact, when a core commit was detected, we considered that its author made a modification to the technical core of the system.

II) Social Network Analysis. Given the list of key developers obtained from the previous step, we investigated whether they (i) belonged to the core of the communication network (mailing list activity), (ii) belonged to the core of the coordination requirements network, (iii) had a high congruence when considering these two networks, or (iv) were top committers. In the following, we briefly describe how we evaluated these four scenarios respectively.

a) Developers in the core of the project's communication network. We collected data from the developers' mailing list using the OSSNetwork tool and built a communication network in the form of an undirected graph. Links were established by detecting developers that contributed to a same mail thread (including the original email sender). For instance, if developer *a* sends an email, and developers *b* and *c* reply to it, then links among all these developers are added to the communication network. Analogously to what was done for the technical network (step I.b), we identified the developers that were in the core of this network by employing the eigenvector centrality measure and doing a quartile analysis.

b) Developers in the core of the coordination requirements network. We generated the project's coordination requirements network using the method proposed by Cataldo *et al.* [5], which relies on the concept of evolutionary dependencies [9]. Such dependencies consist of implicit relationships that are established between software artifacts as they are frequently changed together. This network depicts the set of individuals a developer should coordinate his/her work with (or at least be aware of), since their work tasks share a certain level of interdependency [5, 7]. With the coordination requirements network in hands, we again used eigenvector centrality and a quartile analysis to identify the core of the social network, just as in the previous scenario.

c) Congruence between these two networks. Inspired by the measure of congruence defined Cataldo *et al.* [5], we computed the proportion of social activity that actually occurred (given by the communication network extracted from the mailing list) relative to the social activity that should have taken place (given by the coordination requirements network extracted from the evolutionary dependencies) for each developer. Congruence values thus range between 0 and 1. Such approach for measuring congruence builds on the idea of "fit" from the organizational theory literature [3]. We performed a quartile analysis and the congruence values that were equal to or larger than the third quartile (Q3) were deemed as high.

d) Top contributors. We intend to check whether a small number of developers are responsible for most part of the modifications made to the software system. By using XFlow we computed the top contributors of the Apache Ant project during the studied timeframe, i.e. those developers that made most part of the commits. More precisely, we determined the top committers by analyzing the distribution of *commits per developer*.

III) Comparative Analyses. The final step involved comparing the set of key developers obtained in step I.c with the developers that we identified in the steps II.a, II.b, II.c, and II.d. The results are described in the following section.

5 Results

After collecting the project data by following the aforementioned methods, we divided the results into three groups: the identification of key developers, the analysis of the project's social networks, and the identification of top contributors. In the next subsections we will thoroughly discuss each group of results.

5.1 Identification of Key Developers

As mentioned before, we used JDX to compute the technical network of the codebase corresponding to each Subversion revision of Apache Ant. We then calculated the core of such network and decided whether each revision actually involved a change to the technical core. After that, we calculated the number of core modifications made by each developer. In Table 1 we depict the results we obtained:

Table 1. Developers and Associated Number of Core Modifications to the System

Developer	Number of Core Commits	Delta
ddevienne	0	0
scohen	0	0
umagesh	1	1
conor	1	0
alexeys	2	1
bruce	3	1
jhm	3	0
sbailliez	4	1
kevj	14	10
antoine	25	11
jglick	27	2
jkf	40	13
stevel	77	37
bodewig	118	41
mbenson	172	54
peterreilly	178	6

We sorted the developers according to number of core commits they performed. The thrid column of the table (delta) shows the difference between the number of commits of a developer and his predecessor. The data in this column indicates a first major shift from *jkf* to *stevel* (37). In fact, we notice that approximately 82% of the core commits are performed by a specific group of four developers: *stevel*, *bodewig*, *mbenson*, and *peterreilly*. Therefore, we considered those to be the key developers of Apache Ant during the studied release period.

5.2 The Different Social Networks

In this section, we present the two different social networks we obtained, as well as the measure of congruence for each developer in the Apache Ant project during the studied period.

The Core of the Communication Network. We used OSSNetwork to compute the communication network of the project. Fig. 1 depicts the result we obtained in the form of a graph, in which vertices represents project's developers and edges maps the existence of mail exchanged between two linked vertices.

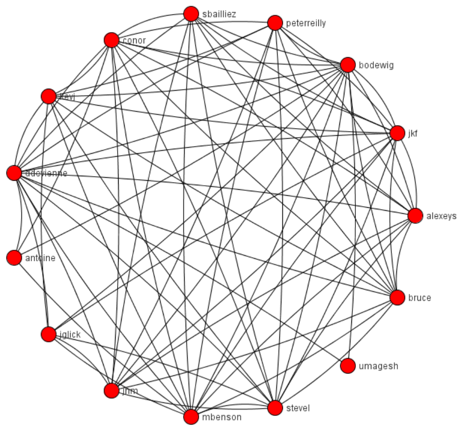

Fig. 1. Communication Network of Apache Ant

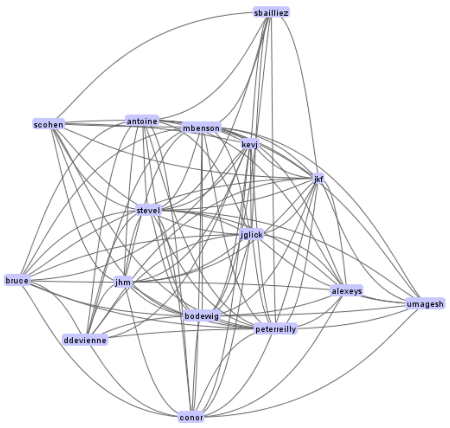

Fig. 2. Coordination Requirements Network of Apache Ant

After that, we employed the Eigenvector centrality measure and the quartile analysis to obtain the set of developers in the core of this network. The results indicated that four individuals are in the core: *bodewig*, *mbenson*, *stevel,* and *jkf*.

The Core of the Coordination Requirements Network. We used XFlow to apply the method proposed by Cataldo *et al.* [5] to calculate the coordination requirements network. Fig. 2 depicts XFlow's graph view of the coordination requirements, where each vertex represents a developer and each edge maps two developers that are likely to coordinate their efforts because the artifacts they are changing are interdependent. After that, analogously to the previous case, we calculated the eigenvector centrality and performed a quartile analysis to obtain the set of developers belonging to the core of this network. The results indicate that a large number of individuals are in the core: *peterreilly*, *bodewig*, *mbenson*, *stevel*, *jkf*, *jglick*, *antoine*, *alexeys*, *jhm*, *sbailliez*, *conor*, *bruce*, *kevj*, and *ddevienne*. Only two developers are not in this list, namely *umagesh* and *scohen*.

Congruence of the Networks. We computed the socio-technical congruence of these two networks for each developer. Fig. 3 depicts the results we obtained. The data shows that the interval of congruence values is large (ranging from 90% to 0%). In a similar fashion to the previous cases, we performed a quartile analysis in order to identify developers with higher congruence. The results we obtained pointed out to four individuals: *ddevienne*, *bodewig*, *kevj*, and *stevel*.

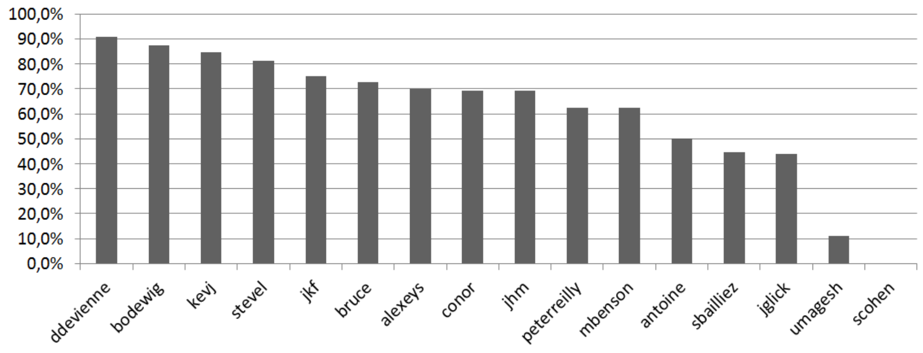

Fig. 3. Socio-technical congruence of the developers

5.3 Top Contributors

We used XFlow and calculated the top contributors of the Apache Ant project during the analyzed period. Fig. 4 depicts the cumulative percentage of the number of commits. According to the data, 4 developers (25% of them) were responsible for 81% of the commits.

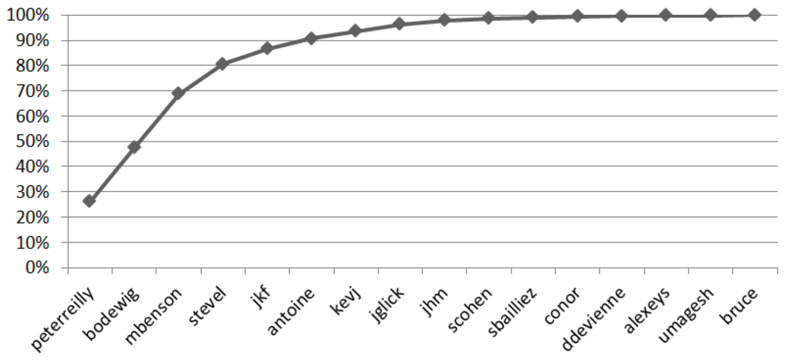

Fig. 4. Cumulative percentage of the number of commits

Therefore, we conclude that the relaxed version of Mockus' hypothesis we defined indeed holds for the Ant project, as most part of the modifications (commits) are made by a small group of developers.

6 Discussion

We start the discussion by illustrating the intersection between the set of key developers and those that (i) are in core of the communication network, (ii) are in the core of the coordination requirements network, (iii) have high socio-technical congruence, and (iv) are top contributors. These results are presented in Table 2.

Table 2. Characterizing key developers

Developer	Key Developer	Core of Communication Network	Core of Coordination Requirements Network	High Congruence	Top Contributors
peterreilly	✓		✓		✓
bodewig	✓	✓	✓	✓	✓
mbenson	✓	✓	✓		✓
stevel	✓	✓	✓	✓	✓
jkf		✓	✓		
jglick			✓		
antoine			✓		
alexeys			✓		
jhm			✓		
sbailliez			✓		
conor		✓	✓		
bruce			✓		
kevj			✓	✓	
ddevienne		✓	✓	✓	
umagesh					
scohen					

Only four key developers were identified, namely: *peterreilly*, *bodewig*, *mbenson*, and *stevel*. Three of these key developers also belonged to the core of the communication network (although such core includes three other developers). This provides evidence that most key developers were also very active in the developers' mailing list during the analyzed period. In relation to the core of the coordination requirements network, all key developers belonged to it. This was somehow expected, since the core of the coordination requirements included 14 of the 16 developers. We think that such core was large due to the inclusive nature of the algorithm used for computing this network: no filters were applied to the evolutionary dependencies, which means that even dependencies between components that occurred only once in the analyzed period are taken into account. We also computed the socio-technical congruence of these two networks for each developer and we noticed that two of the key developers had high congruence. On the other hand, the results also suggest that although *kevj* and *ddeviene* were very communicative (in the sense that they communicated with almost everyone they were required to), they did not work on the technical core very often. Interestingly, the sets of key developers and top contributors are identical (perfect correlation). In fact, by taking a closer look at the volume contribution data, we can see that the set of key developers also heavily contributed to the peripheral areas of the technical network. Finally, only two developers did not show up in any of the considered cases, namely: *umagesh* and *scohen*.

We now answer our research questions in light of the results we obtained. The first question concerned how limited the number of key developers is. As we presented, only four developers (25%) were responsible for approximately 82% of the core modifications. This corroborates our initial expectation that only a few developers would be responsible for making changes to the core. The second question concerned how key developers coordinated their efforts and communicated. After analyzing the developers' mailing list, the coordination requirements network, and the congruence between these two networks (the socio-technical congruence), we noticed that two of the key developers (*bodewig* and *stevel*) were in the core of both networks and also presented a high congruence. The developer *mbenson* was solely in the core of both networks. The developer *peterreilly*, in turn, only appeared in the core of the coordination requirements network. In general, this provides evidence that key developers were often very active in the mailing list (except for *peterreilly*, who was not very active within the project's mailing list). Given the strong connection between software architecture and coordination, we believe that such social interaction help developers coordinate their tasks and keep themselves aware of changes made to the software system. Our third and last research question concerned the contribution volume of key developers. The results showed that key developers were also the ones that contributed the most to the project.

6.1 Threats to Validity

There are some factors that may have influenced the validity of our study.

Construct Validity. Firstly, a common practice in FLOSS development concerns the submission of patches by non-developers interested in helping a particular software project. As these users do not have permission to commit their fixes on the projects' version control system, their contributions are often committed by one of the regular project developers. As a result, this may have introduced some noise in the data used to calculate key developers. Secondly, the webcrawler algorithm employed by OSS-Netwok to parse mailing list data (from HTML pages) makes use of semi-structured webpages as source of information, which is clearly subject to problems due to the lack of rigid rules for participation and participants' identification in the mailing lists. Thirdly, the adoption of eigenvector centrality metric to define core sets on networks might affect our findings. We believe that this measure captures a behavior that seems adequate to our analysis, but other approaches (e.g. k-core or islands) could provide different results. Finally, other thresholds could have been used to determine whether a modification (commit) is core or not.

Internal Validity. Our empirical evidences cover only a single release of the Apache Ant, and it is thus possible that we missed empirical evidence that could be found in other releases of the same project. A more extensive study should be conducted in order to further investigate key developers' characteristics in terms of their social interaction and contributions.

External Validity. Since we studied a single project, we cannot state that these results would remain valid for other projects. In fact, threats to the generalizability of this study are given by the very nature of the employed research design. McGrath [12] states that research methods can be evaluated on three dimensions (generalizability, realism, and precision) and he argues that no method is able to satisfy all dimensions at the same time. In particular, case studies naturally maximize realism, but seldom satisfy generalizability (since they involve a small number of non-randomly selected situations) or precision (because there is a low level of control over influencing factors). Hence, we leverage the realism of our results and conclusions.

7 Conclusion and Future Work

In this paper, we presented a descriptive case study involving Apache Ant. Our goal was to characterize *key developers*, i.e. those developers that effectively evolve the technical core of the project. The reason for studying them is that the access to the technical core of a software project is often restricted to a few developers. In particular, we were interested in answering three research questions that involved investigating (i) *how limited the number of key developers is*, (ii) *how distinct the participation of key developers is (in terms of communication and coordination)*, and (iii) *the contribution volume of key developers*. Our results indicated that only 25% of the developers were classified as key developers. We also showed that key developers were often active in the developers' mailing list and often fulfilled the coordination requirements that emerged from their development tasks. Finally, we noticed that the set of key developers was identical to the set of top contributors.

Our expectations with our findings are that in a long term perspective better characterizing key developers should help researchers understand the process a developer undergoes in order to become a key developer. As these key developers play a crucial role in the project, properly characterizing and identifying them is important in order to better understand the various social processes that often occur within software development. Furthermore, although prior research has tried to understand the process of core-periphery migration on FLOSS projects, the identification of the set of core developers has always been a difficult task that is mostly performed using purely visual methods, which end up posing threats to the validity of these studies and emphasizing the need for more accurate methods.

As future work, we believe that applying our research method to different FLOSS and commercial projects will help verify whether key developers characteristics are similar to those we reported.

Acknowledgements. We thank Steve Abrams for his insightful contribution to the design of the research method, more specifically the usage of eigenvector centrality. Gustavo Oliva receives individual grant from the CHOReOS EC FP7 project. Marco Gerosa receives individual grant from CNPq. Cleidson de Souza was supported by FAPESPA through "Edital Universal N° 003/2008".

References

[1] Balieiro, M., de Júnior, S., de Souza, C.: Facilitating Social Network Studies of Floss Using the Ossnetwork Environment. In: Russo, B., Damiani, E., Hissam, S., Lundell, B., Succi, G. (eds.) Open Source Development, Communities and Quality. IFIP, vol. 275, pp. 343–350. Springer, Boston (2008), `http://dx.doi.org/10.1007/978-0-387-09684-1_31,10.1007/978-0-387-09684-1_31`

[2] Bass, M., Mikulovic, V., Bass, L., Herbsleb, J., Cataldo, M.: Architectural misalignment: An experience report. In: Proceedings of the Sixth Working IEEE/IFIP Conference on Software Architecture, WICSA 2007, p. 17. IEEE Computer Society, Washington, DC (2007), http://dx.doi.org/10.1109/WICSA.2007.12

[3] Burton, R.M., Obel, B.: Strategic Organizational Diagnosis and Design: The Dynamics of Fit, 3rd edn. Information and Organization Design Series. Springer (2003)

[4] Cataldo, M.: Dependencies in geographically distributed software development: overcoming the limits of modularity. Ph.D. thesis, Pittsburgh, PA, USA (2007), aAI3292617

[5] Cataldo, M., Wagstrom, P., Herbsleb, J.D., Carley, K.M.: Identification of coordina-tion requirements: implications for the design of collaboration and awareness tools. In: Hinds, P.J., Martin, D. (eds.) CSCW, pp. 353–362. ACM (2006), http://doi.acm.org/10.1145/1180875.1180929

[6] Conway, M.: How do committees invent. Datamation 14(4), 28–31 (1968)

[7] Costa, J.M., Cataldo, M., de Souza, C.R.: The scale and evolution of coordination needs in large-scale distributed projects: implications for the future generation of collabo-rative tools. In: Proc. of the 2011 Annual Conference on Human Factors in Computing Systems, CHI 2011, pp. 3151–3160. ACM (2011), http://doi.acm.org/10.1145/1978942.1979409

[8] Crowston, K., Wei, K., Li, Q., Howison, J.: Core and periphery in free/libre and open source software team communications. In: Proceedings of the 39th Annual Hawaii International Conference on System Sciences, HICSS 2006, p. 118. IEEE Computer Society, Washington, DC (2006), http://dx.doi.org/10.1109/HICSS.2006.101

[9] Gall, H., Hajek, K., Jazayeri, M.: Detection of logical coupling based on product re-lease history. In: Proceedings of the International Conference on Software Maintenance, ICSM 1998, p. 190. IEEE Computer Society, Washington, DC (1998), http://dl.acm.org/citation.cfm?id=850947.853338

[10] Grinter, R.E.: Systems architecture: product designing and social engineering. SIGSOFT Softw. Eng. Notes 24(2), 11–18 (1999), http://doi.acm.org/10.1145/295666.295668

[11] Levine, J.M., Moreland, R.L.: Progress in small group research. Annual Review of Psychology 41(1), 585–634 (1990), http://arjournals.annualreviews.org/doi/abs/10.1146/annurev.ps.41.020190.003101

[12] McGrath, J.E.: Dilemmatics: The study of research choices and dilemmas. American Behavioral Scientist 25(2), 179–210 (1981), http://abs.sagepub.com/cgi/doi/10.1177/000276428102500205

[13] Mockus, A., Fielding, R.T., Herbsleb, J.D.: Two case studies of open source software development: Apache and mozilla. ACM Trans. Softw. Eng. Methodol. 11, 309–346 (2002), http://doi.acm.org/10.1145/567793.567795

[14] Nakakoji, K., Yamamoto, Y., Nishinaka, Y., Kishida, K., Ye, Y.: Evolution patterns of open-source software systems and communities. In: Proceedings of the International Workshop on Principles of Software Evolution, IWPSE 2002, pp. 76–85. ACM, New York (2002), http://doi.acm.org/10.1145/512035.512055

[15] Newman, M.: Networks: An Introduction, 1st edn. Oxford University Press (2010)

[16] Oezbek, C., Prechelt, L., Thiel, F.: The onion has cancer: some social network analysis visualizations of open source project communication. In: Proceedings of the 3rd International Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development, FLOSS 2010, pp. 5–10. ACM, New York (2010), http://doi.acm.org/10.1145/1833272.1833274

[17] Parnas, D.L.: On the criteria to be used in decomposing systems into modules. Commun. ACM 15, 1053–1058 (1972), http://doi.acm.org/10.1145/361598.361623

[18] Robles, G., Gonzalez-Barahona, J.: Contributor Turnover in Libre Software Projects. In: Damiani, E., Fitzgerald, B., Scacchi, W., Scotto, M., Succi, G. (eds.) Open Source Systems. IFIP, vol. 203, pp. 273–286. Springer, Boston (2006), http://dx.doi.org/10.1007/0-387-34226-5_28,10.1007/0-387-34226-5_28

[19] Robles, G., Gonzalez-Barahona, J.M., Herraiz, I.: Evolution of the core team of developers in libre software projects. In: Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories, MSR 2009, pp. 167–170. IEEE Computer Society, Washington, DC (2009), http://dx.doi.org/10.1109/MSR.2009.5069497

[20] Robson, C.: Real World Research, 2nd edn. John Wiley & Sons (2002)

[21] Santana, F., Oliva, G., de Souza, C.R.B., Gerosa, M.A.: Xflow: An extensible tool for empirical analysis of software systems evolution. In: Proceedings of the VIII Experimental Software Engineering Latin American Workshop, ESELAW 2011 (2011)

[22] de Souza, C., Froehlich, J., Dourish, P.: Seeking the source: software source code as a social and technical artifact. In: Proc. of the 2005 International ACM SIGGROUP Conference on Supporting Group Work, GROUP 2005, pp. 197–206. ACM (2005), http://doi.acm.org/10.1145/1099203.1099239

[23] de Souza, C.R., Quirk, S., Trainer, E., Redmiles, D.F.: Supporting collaborative software development through the visualization of socio-technical dependencies. In: Proc. of the 2007 International ACM Conference on Supporting Group Work, GROUP 2007, pp. 147–156. ACM (2007), http://doi.acm.org/10.1145/1316624.1316646

[24] Souza, C.R., Redmiles, D.F.: On the roles of apis in the coordination of collaborative software development. Comput. Supported Coop. Work 18(5-6), 445–475 (2009), http://dx.doi.org/10.1007/s10606-009-9101-3

[25] Staudenmayer, N.A.: Managing multiple interdependencies in large scale software development projects. Ph.D. thesis, Massachusetts Institute of Technology (1997)

[26] Terceiro, A., Rios, L.R., Chavez, C.: An empirical study on the structural complexity introduced by core and peripheral developers in free software projects. In: Proceedings of the 2010 Brazilian Symposium on Software Engineering, SBES 2010. IEEE Computer Society, Washington, DC (2010), http://dx.doi.org/10.1109SBES.2010.26

[27] Valetto, G., Helander, M., Ehrlich, K., Chulani, S., Wegman, M., Williams, C.: Using software repositories to investigate socio-technical congruence in development projects. In: Proceedings of the Fourth International Workshop on Mining Software Repositories, MSR 2007, p. 25. IEEE Computer Society, Washington, DC (2007), http://dx.doi.org/10.1109/MSR.2007.33

[28] Yin, R.K.: Case study research: Design and methods, 3rd edn. Sage Publications (2003)