Classifying Developers into Core and Peripheral: An Empirical Study on Count and Network Metrics

Mitchell Joblin
Siemens AG
Erlangen, Germany

Sven Apel, Claus Hunsen
University of Passau
Passau, Germany

Wolfgang Mauerer
Siemens AG / OTH Regensburg
Munich / Regensburg, Germany

***Abstract*—Knowledge about the roles developers play in a software project is crucial to understanding the project's collaborative dynamics. In practice, developers are often classified according to the dichotomy of core and peripheral roles. Typically, count-based operationalizations, which rely on simple counts of individual developer activities (e.g., number of commits), are used for this purpose, but there is concern regarding their validity and ability to elicit meaningful insights. To shed light on this issue, we investigate whether count-based operationalizations of developer roles produce consistent results, and we validate them with respect to developers' perceptions by surveying 166 developers. Improving over the state of the art, we propose a relational perspective on developer roles, using fine-grained developer networks modeling the organizational structure, and by examining developer roles in terms of developers' positions and stability within the developer network. In a study of 10 substantial open-source projects, we found that the primary difference between the count-based and our proposed network-based core–peripheral operationalizations is that the network-based ones agree more with developer perception than count-based ones. Furthermore, we demonstrate that a relational perspective can reveal further meaningful insights, such as that core developers exhibit high positional stability, upper positions in the hierarchy, and high levels of coordination with other core developers, which confirms assumptions of previous work.**

I. Introduction

The popular "onion" model—first proposed by Nakakoji et al. [26]—comprises eight roles typically appearing in open-source software projects. These roles extend from passive users of the software, to testers and, active developers. According to this model, there is a clear and intentional expression of the substantial difference in scale between the group sizes fulfilling each role. Quantitative evidence from several empirical studies substantiates this model by showing that the number of code contributions per developer is described by heavy-tailed distributions, which implies that a very small fraction of developers is responsible for performing the majority of work [25], [11], [14]. On the basis of these results, the distinction between different roles of developers is often coarsely represented as a dichotomy comprised of core and peripheral developers [11]. In an abstract sense, *core* developers play an essential role in developing the system architecture and forming the general leadership structure, and they have substantial, long-term involvement [11]. In contrast, *peripheral* developers are typically involved in bug fixes or small enhancements, and they have irregular or short-term involvement [11].

Despite having a substantial understanding about the defining characteristics of core and peripheral developers and recognizing the importance of the interplay between these roles, there is significant uncertainty around core–peripheral operationalizations. A valid and reliable core–peripheral operationalization is crucial for testing empirical evidence of proposed theories regarding collaborative aspects of software development [31], [17]. While several basic operationalizations have been proposed and loosely justified by abstract notions, they may be overly simplistic. For example, one common approach is to apply thresholding on the number of lines of code contributed by each developer [25], but this could result in incorrectly classifying developers who just make large numbers of trivial cleanups. Further evidence suggests that, as a developer moves into a core role, their activity in terms of commit count or lines of code decreases substantially, because they shift their efforts to coordinating the work of others [19]. The major weakness of existing core–peripheral operationalizations stems from the fact that they are primarily based on counting *individual developer* activity (e.g., lines of code, number of commits, number of e-mails sent), which lack any explicit consideration of *inter-developer* relationships. Since many important characteristics of developer roles are concerned with how developers, or their actions, interact with other developers [19], [27], we recognize inter-developer relationships to be of primary importance for the operationalization of developer roles.

The contributions of this work can be summarized by two main achievements. Firstly, we statistically evaluate the agreement between the most commonly used operationalizations of core and peripheral developers by examining data stored in the version-control systems (VCS) and developer mailing lists of 10 substantial open-source projects. Secondly, we establish and evaluate richer notions of developer roles with a basis in relational abstraction. More specifically, we adopt a network-analytic perspective to explore manifestations of core and peripheral characteristics in the evolving organizational structure of software projects, as operationalized by fine-grained developer networks [21], [20]. For evaluation, we performed a survey among 166 developers to establish a ground-truth classification of developer roles to test whether the existing operationalizations and our network-based insights are consistent with respect to each other and valid with respect to developer perception.

In summary, we make the following contributions:

- We statistically evaluate the agreement between classifications of core and peripheral developers generated from commonly used operationalizations—henceforth called *count-based* operationalizations—by studying 10 substantial open-source projects, over at least one year of development, with data from two sources (version-control system and mailing list).
- We conducted a survey among 166 developers to establish a ground truth composed of 982 samples, which we use to evaluate the validity of several core–peripheral operationalizations with respect to developer perception.
- We identify structural and temporal patterns in the project's organizational structure that operationalize core–peripheral roles using network-analysis techniques, referred to as the *network-based* operationalizations.
- We demonstrate that network-based operationalizations exhibit moderate to substantial agreement with the existing count-based operationalizations, but the network-based operationalizations are more reflective of developer perception than the count-based operationalizations.
- We highlight and discuss a number of insights from our network-based operationalizations that are incapable of being expressed by count-based operationalizations, such as positional stability, hierarchical embeddings, and interaction patterns between core and peripheral developers.

All experimental data and source code are available at a supplementary Web site.[1]

II. Background & Related Work

Researchers have examined the core and peripheral developer roles from two distinct perspectives: from a social perspective, by studying communication and collaboration patterns [8], [23], [25], [14], [19], and from a technical perspective, by studying patterns of contributions of developers to technical artifacts [11], [25], [32], [18], [14], [19]. Regarding social characteristics, core developers play a central role in the communication and leadership structure [8] and have substantial communication ties to other core developers, especially in projects with a small developer community (10–15 people) [23], [25]. Regarding technical characteristics, core developers typically exhibit strong ownership over particular files that they manage, they often have detailed knowledge of the system architecture, and they have demonstrated themselves to be extremely competent [11], [25], [32], [18]. In contrast, peripheral developers are primarily involved in identifying code-quality issues and in proposing fixes, while also participating moderately in development-related discussions [25]. As the roles of developers are not static, prior research has also investigated temporal characteristics of core and peripheral developers in terms of the advancement process to achieving core-developer status. Advancement is typically merit-based and often involves long-term, consistent, and intensive involvement in a project [25], [10], [33], [18], [19].

Many of the aforementioned studies applied empirical methods based on interviews, questionnaires, personal experience reports, and manual inspections of data archives to identify characteristics of core and peripheral developers. An alternative line of research has attempted to operationalize the roles of core and peripheral developers using data available in software repositories, such as version-control systems [25], [32], [30], [29], [27], [12], bug trackers [11], and mailing lists [27], [3]. By operationalizing the notion of core and peripheral developers, these studies have taken important steps towards gaining insight that is not attainable with (more) manual approaches, including evaluating and basing conclusions on results from hundreds of projects [10].

Despite the usage of numerous operationalizations by researchers [11], [25], [32], [30], [29], [19], we have very limited knowledge about their validity, though. There is a reasonable cause for concern that some corresponding metrics are overly simple: Most operationalizations are single-dimension values that represent the developer's activity level (e.g., the number of commits made), with a corresponding threshold based on a prescribed percentile. A commonly used approach is to count the number of commits made by each developer, and then to compute a threshold at the 80% percentile. Developers that have a commit count above the threshold are considered core, developers below are considered peripheral [11], [25], [32], [30], [29]. This threshold was rationalized by observing that the number of commits made by developers typically follows a Zipf distribution (which implies that the top 20% of contributors are responsible for 80% of the contributions) [11]. Following a similar direction, Mockus et al. found empirical evidence for Mozilla browser and Apache Web server that a small number of developers are responsible for approximately 80% of the code modifications [25]. However, in a replication study performed on FreeBSD, the results indicated that a set of "top developers" are responsible for approximately 80% of the changes, but this group does not coincide well with the elected core developer group [14]. Further attempts have been made to investigate the difference between core and peripheral developers by using basic social-network centrality metrics and a corresponding threshold [27], [12], [3]. In these cases, developer networks have been constructed on a dyadic domain of either mutual contributions to mailing-list threads or source-code files.

While many approaches exist to classify developers into core and peripheral, no substantial evidence has been accumulated to evaluate the validity and consistency of these different operationalizations. Crowston et al. [11] investigated three operationalizations of core and peripheral developers, but they focused only on bug-tracker data and neglected code authorship entirely. Olivia et al. [27] dedicated attention on developing a more detailed characterization of so-called "key developers", which is similar to the core-developer dichotomy. They investigated mailing lists and version-control systems with three operationalizations to classify developers as core or

[1] http://siemens.github.io/codeface/icse2017/

peripheral. Their results indicate that there is some evidence of agreement between the different operationalizations, but this was only shown for one release of a single small project with only 16 developers, in total, and 4 core developers. We improve over the state of the art by considering a larger and more diverse set of projects with larger developer communities, by using more metrics, and by analyzing, at least, one year of development, to evaluate the temporal stability of our results. Additionally, we use developer-network models that have been validated in prior studies [21], [4].

III. COUNT-BASED OPERATIONALIZATIONS

Based on a review of the existing literature, we have identified three variations of count-based operationalizations of core–peripheral roles [25], [32], [30], [29], [27], [12], [11], [3]. In the literture, metrics are used with a percentile threshold to define a dichotomy composed of core and peripheral developers. We apply the standard 80th percentile threshold, because of its wide use and its justification based on the data following a Zipf distribution (see Section II). Two operationalizations capture technical contributions to the version-control system and one captures social contributions to the developer mailing list.

Commit count is the number of commits a developer has authored (merged to the master branch). A commit represents a single unit of effort for making a logically related set of changes to the source code. Core developers typically make frequent contributions to the code base and should, in theory, achieve a higher commit count than peripheral developers.

Lines of code (LOC) count is the sum of added and deleted lines of code a developer has authored (merged to the master branch). Counting LOC, as it relates to developer roles, follows a similar rationale as commit count. As core developers are responsible for the majority of changes, they should reach higher LOC counts than peripheral developers. A potential source of error is that developers writing inefficient code or changing a large number of lines with trivial alterations (e.g., whitespace changes) could artificially affect the classification.

Mail count is the number of mails a developer contributed to the developer mailing list. Core developers often posses in-depth technical knowledge, and the mailing list is the primary public venue for this knowledge to be exchanged with others. Core developers offer their expertise in different forms: making recommendations for changes, discussing potential integration challenges, or providing comments on proposed changes from other developers. Typically, peripheral developers ask questions or ask for reviews on patches they propose. Core developers often participate more intensively and consistently and have greater responsibilities than peripheral developers, in general. This should result in core developers making a large number of contributions to the mailing list. This is still only a very basic metric because a developer answering many questions and one asking many questions will appear to be similar, and there is no inter-developer information, so who is speaking with whom or with how many people is completely ignored.

Each of the above metrics has a foundation rooted in our current empirical understanding of the characteristics of core and peripheral developers, but in the end, they are all relatively simple abstractions of a potentially multifaceted and complex concept [14]. A comparison between the resulting classification of developers from these different metrics will provide valuable insights into whether systematic errors exist in these count-based operationalizations, which we perform in Section VI-A. However, as the focus of these metrics is still only to assign developers exclusive membership to one of two unordered sets—without relational information between sets or within the sets—the insights offered by the classification are of limited practical value. To address this shortcoming, we propose a relational view on developer coordination and communication as the basis for developer-role operationalizations.

IV. A NETWORK PERSPECTIVE

A developer network is a relational abstraction that represents developers as nodes and relationships between developers as edges. The promise of a network perspective is greater practical insights concerning the organizational and collaborative relationships between developers [8], [24], [9], [12], [4]. But to what extent can this promise be fulfilled? So far, we know that developer networks, when carefully constructed on version-control-system and mailing-list data, can be both accurate in reflecting developer perception and reveal important functional substructure, or communities, with related tasks and goals [21], [4]. What can be elicited from developer networks regarding the core–peripheral dichotomy has not yet been greatly explored, especially in comparison to non-network-based approaches, which is our intention in this work. Practical opportunities for network insights are, for example: identifying core developers that are overwhelmed by the peripheral developers they need to coordinate with; structural equivalence (that is two nodes with the same neighbors) could reveal which core developers have similar knowledge or technical abilities, which helps to determine appropriate developers for sharing or shifting development tasks; structural holes between core developers may indicate deteriorating coordination; or a single globally central core developer may indicate an important organizational risk.

A. Network Model

We now present the details of our network-analytic approach for analyzing data from version-control systems and mailing lists to examine relational characteristics of core and peripheral developers. Intuition and prior research led us to the conclusion that the role a developer fulfills can change over time [18]. For this reason, we analyze multiple contiguous periods over one year of a project in question using overlapping analysis windows. Each analysis window is three months in length, and each subsequent analysis period is separated by two weeks [20]. We chose three-month analysis windows, because it has been shown that, beyond this window size,

the development community does not change significantly, but temporal resolution in their activities is lost [24].

E-mail networks: For a given project, we download the mailing lists archives either from *gmane* using *nntp-pull* or directly from the project's homepage to obtain an *mbox* formatted file containing all messages sent to the mailing list. Most projects have different mailing lists for different purposes. We consider only the primary mailing list for development-related discussions. We apply several preprocessing steps to remove duplicated messages, normalize author names, and organize the mails into threads using the *Message-IDs* and *In-Reply-To-IDs* [15]. Furthermore, we decompose the *From* lines of each mail into a ⟨name, e-mail address⟩ pair. In some cases, only an e-mail address or only a name is possible to recover, and this can present issues with identifying all e-mails that a single person sent. To resolve multiple aliases to a single identity, we use a basic heuristic approach similar to the one proposed by Bird et al. [2]. Despite the potential problems regarding author-name resolution—as developers accumulate valuable credibility through contributions to the mailing list—it is counterproductive for highly active individuals to use multiple aliases and conceal their identity. To construct a network representation of developer communication, we apply the standard approach, where edges are added between individuals who make subsequent contributions to a common thread of communication [2].

Version-control-system networks: Data in version-control systems are organized in a tree structure composed of commits. We analyze only the main branch of development, as a linearized history, by flattening all branches merged to master. Furthermore, we analyze only the *authors* of commits, not the committer (which are expressed differently in Git), and attribute the commit to a unique individual using the same aliasing algorithm as for the mailing-list data. We count lines of code for each commit based on diff information, where the total line count is the sum of added and deleted lines. The network representation of developer activities in the version-control system is constructed using fine-grained *function*-level information, which was observed to produce authentic networks that agree with developer perception [21]. In this approach, changes are localized to function blocks using source-code structure to identify when two developers edit interdependent lines of code. We enhance the network with semantic-coupling relationships between functions, which has shown to also reflect developer perception of artifact coupling [1]. The semantic relationships are identified by making use of the domain-specific words that are embedded in the textual content of the implementation (e.g., variable and function identifiers) [20].

B. Core and Peripheral Developers in Developer Networks

In Section II, we discussed the expectation that characteristics of core and peripheral developers should manifest in ways that transcend the count-based operationalizations introduced in Section III—an expectation that is also supported by a survey among 166 open-source developers (see Section VI-E). Next, we introduce five network-based operationalizations that are rooted in the structure and evolution of developer networks (see Section IV-A).

Degree centrality aims at measuring local importance. It represents the number of ties (edges) a developer has to other developers [7]. As essential members of the leadership and coordination structure, core developers associate with other core members and with peripheral developers that require their technical guidance. Peripheral developers are likely involved in only a small number of isolated changes and thus have only a limited number of interactions with other members of the development community. The expectation is that core developers then have a larger degree than peripheral developers.

Eigenvector centrality is a global centrality metric that represents the expected importance of a developer by either connecting to many developers or by connecting to developers that are themselves in globally central positions [7]. Since core developers are critical to the leadership and coordination structure, we expect them to occupy globally central positions in the developer network.

Hierarchy is present in networks that have nodes arranged in a layered structure, such that small cohesive groups are embedded within large, less cohesive groups. In a hierarchical network, nodes with high degree tend to have edges that span across cohesive groups, thereby lowering their clustering coefficient [28]. Prior work has shown that developers tend to form cohesive communities [21], and we expect core developers to play a role in coordinating the effort of these communities of developers. If this is true, then core developers should have a high degree and low clustering coefficient, placing them in the upper region of the hierarchy, while peripheral developers should exhibit a comparatively low degree and high clustering coefficient, placing them in the lower region of the hierarchy.

Role stability is a temporal property of how developers switch between roles. For this reason, we investigate the patterns of developers' transitions through different roles by observing changes in the corresponding developer network over time. As core developers typically attain their credibility through consistent involvement and often have accumulated knowledge in particular areas of the system over substantial time periods (see Section II), we expect their stability in the developer network to be higher than for peripheral developers. We operationalize developer stability by estimating the probability that a developer in a given role leaves the project by not participating for, at least, three months. For each developer the role during each development window is determined using the degree-centrality operationalization. The time-ordered sequence of roles for each developer is then used in a maximum-likelihood estimation to solve for each state transition parameter (e.g., the probability that a core developer transitions to a peripheral role) [5].

Core–peripheral block model is a formalization, proposed in the social-network literature, that captures the notion of core–periphery structure based on an adjacency-matrix representation. The block model specifies the core–core region of the matrix as a 1-block (i.e., completely connected), the core–

peripheral regions as imperfect 1-blocks, and the peripheral–peripheral region as a 0-block [34]. Intuitively, this model describes a network as a set of core nodes, with many edges linking each other, surrounded by a loosely connected set of peripheral nodes that have no edges connecting each other. Of course, this idealized block model is rarely observed in empirical data [6]. Still, we are able to draw practical consequences from this formalization by estimating the edge presence probability of each position to test if core and peripheral developers (operationalized by degree centrality) occupy core and peripheral network positions according to this block model. From the block model, one can mathematically reason that the probability of observing an edge in each block is distinct and related according to $p_{\text{core–core}} > p_{\text{core–periph}} > p_{\text{periph–periph}}$ [34]. This model aligns with empirical data that indicate that core developers are typically well-coordinated and are expected to be densely connected in the developer network [25]. Since peripheral developers often rely on the knowledge and support of core developers to complete their tasks, it follows that peripheral developers often coordinate with core developers, and only in rare cases would we expect substantial coordination between peripheral developers. This expected behavior aligns very well to the formalized notion of core–periphery positions from social-network analysis.

V. Empirical Study

A. Subject Projects

We selected ten open-source projects, listed in Table I, to study the core–peripheral developer roles. We specifically chose a diverse set of projects to avoid biasing the results. The projects vary by the following dimensions: (a) size (source lines of code from 50KLOC to over 16 MLOC, number of developers from 15 to 1000), (b) age (days since first commit), (c) technology (programming language, libraries), (d) application domain (operating system, development, productivity, etc.), (e) development process employed. Developers of the project referred to as Project X have requested that their project name remains anonymous.

B. Research Questions

While each of the approaches for classifying core and peripheral developers is inspired by common abstract notions rooted in empirical results, it has not been shown that the approaches agree. It may be the case that they capture orthogonal dimensions of the same abstract concept, which gives rise to our first research question:

RQ1: Consistency—*Do the commonly applied operationalizations of core and peripheral developers based on version-control-system and mailing-list data agree with each other?*

Compared to the extent of our knowledge regarding the characteristics of core and peripheral developers, existing count-based operationalizations are relatively simple. Since core developers often have strong ownership over particular files and play a central role in coordinating the work of others on those artifacts [8], [25], [19], we would expect core developers to differ, in a relational sense, from peripheral developers in how they are embedded in the communication and coordination structure. Furthermore, as core developers typically achieve their status through long-term and consistent involvement [18], we expect their temporal stability patterns to differ from peripheral developers.

RQ2: Positions & Stability—*Do the differences between core and peripheral developers manifest in relational terms within the communication and coordination structure with respect to their positions and stability?*

The utility offered by an operationalization is limited by the extent to which the operationalization is able to accurately capture a real-world phenomenon. So far, it is unclear to what extent the core–peripheral operationalizations reflect developer roles as seen by their peers. We explore whether relational abstraction, as in the network-based operationalizations, improves over the count-based operationalizations by more accurately reflecting developer perception through explicit modeling of developer–developer interactions.

RQ3: Developer Perception—*To what extent do the various count-based and network-based operationalizations agree with developer perception?*

C. Hypotheses

The existing count-based operationalizations of core and peripheral developers discussed in Section III claim to be valid measures, and if this is a matter of fact, we expect to reach consistent conclusions about whether a given developer is core or peripheral. Due to finite random sampling and sources of noise, we expect imperfect agreement between two operationalizations even if they are consistent in capturing the same abstract concept. However, if the operationalizations are consistent, the level of agreement should be significantly greater than the case of random assignment of developer roles. Our null model for zero agreement is the amount of agreement that results from two operationalizations that assign classes according to a Bernoulli process.[2] To operationalize agreement between two binary classifications (core or peripheral) of a given set of developers, we use Cohen's kappa, $\kappa = (p_o - p_e)/(1 - p_e)$, where p_o is the number of times the two classifications agree on a role of a developer, divided by the total number of developers and where p_e is the expected probability of agreement when there is random assignment of roles to developers but the proportion of each class is maintained. Cohen's kappa is more robust than simple percent agreement because it incorporates the effect of agreement that occurs by chance [22]. This characteristic is particularly important in our case since the frequency of roles is highly asymmetric as the majority of developers are peripheral and only a small fraction are core. The ranges for Cohen's kappa and corresponding strengths of agreement are: 0.81–1.00 al-

[2] A Bernoulli process generates a sequence of binary-valued random variables that are independent and identically distributed according to a Bernoulli distribution. The process is essentially simulating repeated coin flipping.

TABLE I: Overview of subject projects

							Edge Probabilities			Hierarchy	
Project	Domain	Lang	Devs	SLOC	Commits	Date	C–C	C–P	P–P	Rho[1]	p value
Project X	User	C/++, JS	826	10M	276K	2015-12-05	9.75e-02	4.19e-03	2.70e-03	-0.552	5.51e-33
Django	Devel	Python	100	430K	41K	2015-12-06	2.95e-01	9.09e-03	3.08e-03	-0.812	1.28e-06
FFmpeg	User	C	103	1M	78K	2015-11-08	5.50e-01	2.44e-02	5.16e-03	-0.725	7.10e-06
GCC	Devel	C/++	122	7.5M	144K	2015-11-03	4.07e-01	1.84e-02	1.01e-02	-0.646	1.12e-04
Linux	OS	C	1467	18M	637K	2015-12-05	2.39e-02	5.93e-04	3.60e-04	-0.689	6.06e-62
LLVM	Devel	C/++	180	1.1M	62K	2015-11-02	7.80e-01	5.54e-02	2.62e-02	-0.778	8.72e-24
PostgreSQL	Devel	C	17	1M	40K	2015-12-05	1.00e+00	1.62e-01	5.13e-02	-0.871	1.31e-03
QEMU	OS	C	134	1M	43K	2015-11-02	3.20e-01	1.95e-02	1.16e-02	-0.756	4.76e-07
U-Boot	Devel	C	142	1.3M	35K	2015-11-01	2.00e-01	7.59e-03	4.20e-03	-0.728	8.27e-05
Wine	User	C	62	2.8M	110K	2015-11-06	3.46e-01	2.91e-02	1.28e-02	-0.832	1.04e-05

[1] Spearman's correlation coefficient

most perfect, 0.61–0.80 substantial, 0.41–0.6 moderate, 0.21–0.40 fair, 0.00–0.20 slight, and < 0.00 poor [22].

H1—*Existing count-based operationalizations of core and peripheral developers based on version-control-system and mailing-list data are statistically consistent in classifying developer roles.*

The abstract notion of core and peripheral developers discussed in Section II emphasizes the multitude of ways the two groups differ (e.g., contribution patterns, knowledge, level of engagement, organization, responsibility, etc.). While existing operationalizations of core and peripheral developers are primarily based on simple metrics of counting high-level activities of developers, these metrics largely ignore the richness in the definition of core and peripheral roles. In particular, the dimension of time is largely ignored, though time plays a central role in the developer-advancement process [18].

H2—*The well-known abstract characteristics of core developers will manifest as distinct structural and temporal features in the corresponding developer network: Core developers will exhibit globally central positions, relatively high positional stability, and hierarchical embedding.*

As core developers form the primary coordination structure, we expect to observe: many edges in the developer network between core developers, less edges between core and peripheral developers, and even fewer edges between peripheral developers. We investigate this hypothesis in terms of preferences between the groups to associate based on the probability of an edge occurring between them according to the core–peripheral block model (see Section IV-B).

H3—*Core developers have a preference to coordinate with other core developers; peripheral developers have a preference to coordinate with core developers instead of other peripheral developers.*

We expect developer networks to reveal core and peripheral developers, albeit in a richer representation, with comparable precision to the currently accepted operationalizations. More specifically, we expect developer networks capture the core–peripheral property to an equally high standard as the currently accepted operationalizations; any disagreement should be on the order of the discrepancy between existing operationalizations.

H4—*The core–peripheral decomposition obtained from developer networks will be consistent with the core–peripheral decomposition obtained from the prior accepted operationalizations. The discrepancy in agreement will not exceed the amount observed between the existing operationalizations.*

As the count-based operationalizations appear to reasonably capture simple aspects of developer roles, we expect a certain level of agreement between these operationalizations and developer perception. In the case of the network-based operationalizations, we expect even higher agreement with developer perception since the relational abstraction explicitly captures developer–developer interactions, which are neglected by the count-based operationalizations.

H5—*Count-based operationalizations agree with developer perception, but network-based operationalizations exhibit higher agreement.*

D. Developer Perception

To establish a ground-truth classification of developer roles, we designed an online survey in which we asked developers to report the roles of developer's in their project according to their perception. The goal of acquiring these data is to test whether the core–peripheral operationalizations are valid with regard to developer perception (not only to other operationalizations). A sample of the survey instrument can be found at the supplementary Web site.

We recruited participants for the study from the version-control-system data of our ten subject projects by identifying the e-mail addresses of individuals that made a commit within the three months prior to the survey date (see Table I). This was to ensure that the selected developers have current knowledge of the project state, so that their answers are temporally consistent with our analysis time frame. One subject project, GCC, was excluded from the survey because the developer e-mail addresses are not available in the version-control system. For the remaining 9 projects, we sent recruitment e-mails to 3369 addresses of which 166 elicited a complete response. In total, we obtained 982 role classifications. The distribution of responses from the projects was 41% for Linux, 7% for

Django, 8% for QEMU, 15% for LLVM, 8% for PostgreSQL, 13% for Wine, and 7% for FFmpeg. One explanation for the low response rate is that the e-mail addresses in the version-control system are often inactive or unreachable.

The survey includes two primary sections: The first section contains questions that require the developers to self-report their role in the project (core or peripheral) and to provide a textual description of the nature of their participation. This question is useful for identifying potential sampling-bias problems and to determine if developers' self-reported role is consistent with the answers provided by their peers. The second section includes a list of 12 developers, identified by name and e-mail address, sampled from their specific project. For each developer appearing in the list, the respondent was asked to provide a classification of the developer's role. Appropriate options are also available if the respondent did not know the developer in question or was unsure of the role. We applied the following sampling strategy to select the list of twelve developers: Five developers were randomly selected from the core group and five from the peripheral group, classified according the the commit count-based operationalization (see Section III). The remaining two developers were randomly selected from the direct neighbors, in the developer network, of the survey participant. We chose to use neighbors because it is likely that neighbors work directly together and are aware of each other's roles.

VI. Results

We now present the results of our empirical study and address the five hypotheses described in Section V-C. For practical reasons, we are only able to present figures for a single project that is representative of the general results. Please refer to the supplementary Web site for the remaining project figures.

A. RQ1: Consistency of Count-Based Operationalizations

To address H1, we compute the pairwise agreement between all count-based metrics for a given project. For this purpose, we analyze each subject project in a time-resolved manner using a sliding-window approach (see Section IV) to generate time-series data that reflect the agreement for a particular three-month development period. An example time series is shown in Figure 1 for QEMU. While being only one project, the insights are consistent with the results from the other projects. The figure illustrates the agreement for Cohen's kappa, and we see that, for all comparisons, the agreement is greater than fair (e.g., greater than 0.2), which significantly exceeds the level of agreement expected by chance (see Section V-C). This is evidence that the different count-based operationalizations do not contradict each other. For operationalizations that are based on the same data source (i.e., version-control system), we typically see substantial agreement (0.61–0.8). One reason for the lower cross-archive agreement could be due to problems of multiple aliases, which will be discussed in detail in Section VII. Another interesting result is that the agreement is relatively stable over time, which

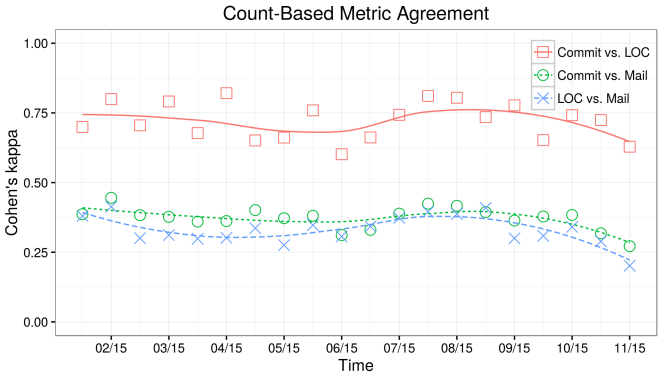

Fig. 1: QEMU time series representation of pairwise agreement between count-based operationalizations. The data indicate that agreement is fair to substantial and is temporally stable (i.e., mean and variance are time invariant).

is again visible in Figure 1 for QEMU. More specifically, the arithmetic mean and variance do not significantly change over time—a property referred to as "wide-sense stationary" in the time-series analysis literature [16]. This feature of the data is a testament to the validity of the operationalizations, as we would not expect the agreement between operationalizations to change drastically from one development window to the next. The wide-sense stationary property is also important because it permits us to aggregate the data by averaging over the time windows to attenuate noise and generate more concise overviews without sacrificing scientific rigor or interpretability of the result.

> Overall, the results demonstrate that the count-based operationalizations largely produce consistent results regarding the classification of developers into core and peripheral groups. We therefore *accept H1*.

B. RQ2: Core and Peripheral Developers in Developer Networks

Hierarchy: In a hierarchical network, nodes at the top of the hierarchy have a high degree and low clustering coefficient; nodes at the bottom of the hierarchy have a low degree and high clustering coefficient [28]. If hierarchy exists in a developer network, we should see mutual dependence between the clustering coefficient and the degree of nodes in the network [28]. The hierarchical relationship for QEMU is shown in Figure 2; there is an obvious dependence between the node degree and clustering coefficient. Nodes with a high degree are seen to exclusively have very low clustering coefficient and are indicative of core developers according to Section IV-B; low degree nodes have consistently higher clustering coefficients and are indicative of peripheral developers. For the remaining projects, the scatter plots are available on the supplementary Web site; here, we illustrate the relationship in a more compact form in terms of Spearman's correlation coefficient between clustering coefficient and degree (see Table I "Hierarchy"). We see that, for all projects, there is a strong negative correlation, indicating that the developers are indeed arranged hierarchically. In Sections VI-C and VI-D, we will see if

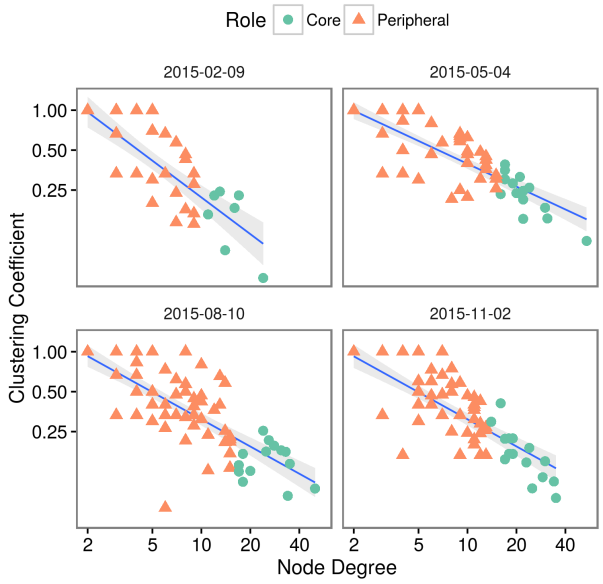

Fig. 2: QEMU's developer hierarchy during four development periods. The linear dependence between clustering coefficient and degree expresses the hierarchy. Core developers should appear clustered at the top of the hierarchy (bottom right region) and peripheral developers at the bottom of the hierarchy (upper left region)

a developer's position in the hierarchy is an organizational manifestation of their particular role.

Stability: Developers who fulfill a particular role within a project and who maintain participation over subsequent development periods are defined to be stable (see Section IV-B). We study this characteristic by examining the developers' transitions from one state to another (e.g., core to peripheral) in a time-resolved manner. The result of examining the developer transitions over one year of development for QEMU are shown in Figure 3. In this figure, the transition probabilities between developer states are shown in the form of a Markov chain. The primary observation is that developers in a core state are substantially less likely to transition to the absent state (i.e., leave the project) or isolated state (i.e., have no neighbors in the developer network by working exclusively on isolated tasks), in comparison to developers in a peripheral state. So, the core developers represent a more stable group than peripheral developers.

Core–periphery block model: The core–periphery block model describes the core and peripheral groups, formalized as positions in a network, as a particular two-class partition of nodes (see Section IV-B). To test whether our empirical data are plausibly described by the core–periphery block model, we must compute the edge-presence probabilities for core–core, core–peripheral, and peripheral–peripheral edges. If the edge-presence probabilities are arranged according to, $p_{\text{core–core}} > p_{\text{core–periph}} > p_{\text{periph–periph}}$, then we can conclude that core developers constitute the most coordinated developers in the project, peripheral developer coordinate primarily with core developers, and peripheral developers rarely coordinate

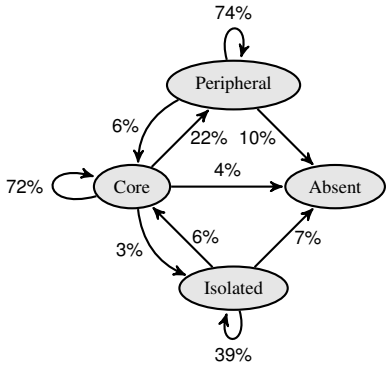

Fig. 3: Developer-group stability for QEMU shown in the form of a Markov Chain. A few less important edges have been omitted for visual clarity.

with other peripheral developers. This provides an example of a relational perspective that captures intra- and inter-relational information (see Section IV-B).

The edge-presence probabilities for all projects are shown in Table I (column "Edge Probabilities"). In all projects, the inequality holds, indicating that the model plausibly describes our projects. The edge-presence probability for core–core has a mean value of 4.02×10^{-1}, for core–peripheral edges it is significantly lower with a mean value of 3.30×10^{-2}, and the peripheral–peripheral edge probability is lower yet with a mean value of 1.28×10^{-2}. The interpretation is that peripheral developers are twice as likely to coordinate with core developers as opposed to other peripheral developers.

Two projects are noteworthy outliers, but are still described by the core–periphery block model: Linux and PostgreSQL. For Linux, the edge-presence probabilities are notably lower in all cases, and the difference in scale between core–core edge probabilities and the others is two orders of magnitude. In the case of PostgresSQL, we see an outlier in the opposite direction. The core–core edge probability is 1, notably higher than for all other projects, much like core–peripheral edges. It is interesting that both of these projects are also outliers in terms of the size of the developer community: Linux is much larger than most projects (1510 developers), PostgreSQL is much smaller (18 developers). From this result, it appears that the scale of a project influences how likely it is for two developers to coordinate, and this influence has a greater effect on the coordination of peripheral developers.

Overall, the network-based operationalizations illustrate clear manifestations of core and peripheral developer roles that agree with the abstract characteristics established by earlier empirical work. We also found evidence in terms of the core–peripheral block model that developer roles imply specific coordination preferences. On the basis of these results, we *accept H2 and H3.*

C. Agreement: Network-Based vs. Count-Based

So far, our results have provided evidence that the count-based operationalizations produce consistent classifications of developers, which is a testament to their validity, and that

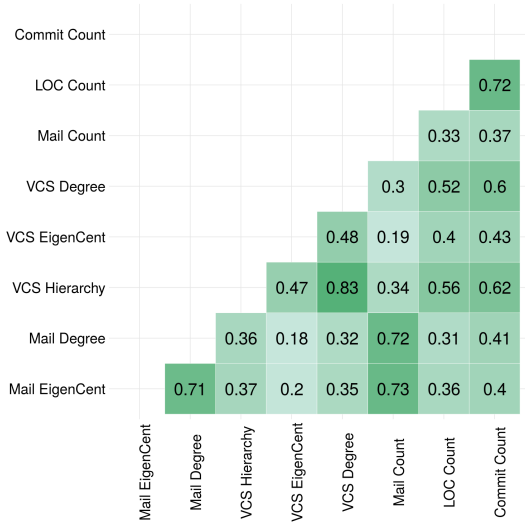

Fig. 4: Time-averaged agreement in terms of Cohen's kappa for QEMU. The pairwise agreement is shown for the count-based and network-based operationalizations

developer networks exhibit specific characteristics that are indicative of core and peripheral developer roles. Next, we present the results to relate the network-based to the count-based operationalizations for identifying core and peripheral developers. We approach this evaluation again using Cohen's kappa by averaging the level of agreement over one year of development. QEMU is used as an example project and the pairwise agreement for each operationalization is illustrated in Figure 4. The stability and core–periphery block-model operationalizations do not show up explicitly since they are derived from degree centrality.

In general, the level of agreement always exceeds 0, which indicates that the strength of agreement between all operationalizations significantly exceeds what is expected by chance. The rows/columns beginning with "VCS" are based on data stemming from the version-control system, and those with "Mail" are based on the mailing list. We again see that agreement between operationalizations defined on same data source typically have substantial agreement (0.6–0.8).

Overall, the results indicate that the network-based and count-based operationalizations are mostly consistent. While the agreement is imperfect, the results show that the divergence from perfect agreement is similar what is seen among the count-based operationalizations. We therefore *accept H4*.

D. RQ3: Developer Perception vs. Network-Based and Count-Based Operationalizations

To establish a ground-truth classification of developers based on the perception of our survey participants, we computed the number of core and peripheral votes for each developer from the survey responses (see Section V-D). For each developer, we chose the role with the highest number of votes as the ground truth and, if the count was equal, the developer was removed. Upon inspection of the responses, we found that they were largely consistent regarding a given developer's role.

TABLE II: Agreement with developer perception

		Cohen's kappa	p value
Counts	Commit Count	0.387	3.12e-06
	LOC Count	0.355	1.91e-05
	Mail Count	0.421	2.08e-05
Networks	VCS Degree	0.465	4.48e-08
	VCS Hierarchy	0.437	2.22e-07
	VCS EigenCent	0.404	1.74e-06
	Mail Degree	0.497	8.23e-07
	Mail EigenCent	0.427	1.26e-05

The results of comparing both the count and network-based operationalizations to the ground-truth classification are shown in Table II. Agreement was computed for 163 ground-truth samples provided by the survey participants. Three participant responses were eliminated because they were incomplete.[3]

The nominal agreement values, in terms of Cohen's kappa, exceed 0 indicating that all operationalizations agree with developer perception significantly more than what is expected by chance (see Section V-C). The highest and second highest agreement is seen in the node-degree metric for the E-mail network and VCS network, respectively. The lowest agreement is seen for the count-based version-control-system metrics (Commit and LOC count). In general, all network-based operationalizations agree better (albeit in some cases only slightly) with developer perception than the basic version-control-system count-based metrics. Focusing on the comparison between different data archives, the agreement for the mailing lists metrics have even greater agreement than the corresponding version-control-system metric. However, network-based metrics always outperform the count-based metrics when the data source is the same.

In general, the mailing list is most accurate in capturing characteristics that reflect developer perception of roles. However, in many projects communication archives are not available, and in this case, a network perspective on version-control system data can closely resemble the insights (regarding developer roles) provided by the communication archive. Overall, we see that a network perspective always improves the agreement with developer perception over the simpler count-based operationalizations. To this end, *we accept H5*.

E. Support for Relational Perspective

In addition to providing data for testing our hypotheses, the developer survey provides additional evidence for and insights into the usefulness of a relational perspective on developer roles. Our survey results suggest that developer roles are often defined in terms of differences in the mode of interaction between developers. For example, one developer wrote "*core maintainers participate in discussions on areas outside the ones that they maintain*". Only a relational perspective is able to capture this view, for example, in terms of core developers having a higher degree than peripheral developers,

[3]The survey response data are available at the supplementary Web site.

because they interact with developers working in areas that are distinct from the ones that they maintain. In the same vein, core developers are likely to occupy upper positions in a hierarchy, as they provide coordination bridges between the peripheral developers that have a comparatively narrow focus. Another core developer mentioned, "*I may not be contributing as much as I did in past years, but I am still active and available to answer questions from and provide guidance to other developers.*" Again, the developer has emphasized their role based on a mode of interaction with other developers. Another survey participant commented: "*The Wine project has lots of committers and a very loose structure. It's very hard to know who does what.*" A relational view on the global organizational structure has practical value to support this kind of developer awareness that is currently missing. Beside static network properties, we argue that the temporal dimension is needed to accurately operationalize developers roles, which is also supported by survey responses: "*The boundaries are fuzzy and can change over time — sometimes I'm a core developer on libvirt, while at the present I'm only a peripheral developer*" or "*I tend to classify contributors as regular opposed to occasional.*" This is especially important as count-based operationalizations do not capture temporal relationships.

VII. Threats to Validity

Construct Validity: Quantifying the extent to which the operationalizations of developer roles represent the real world is one of the primary contributions of this work. We used the concept of mutual agreement as a testament to the validity of the operationalizations, however, one explanation for observing mutual agreement could be that all the operationalizations consistently reach the same wrong conclusion. While this would be a rather improbable explanation, we carried out a developer survey to provide additional evidence for that the operationalizations are valid.

For the network-based operationalizations, we used developer networks and network-analysis techniques to establish a relational basis for studying core and peripheral developers. This poses the threat that the networks and metrics do not accurately capture reality. This threat is minor as there is already evidence indicating that both the networks and the metrics are authentic in reflecting developer perception [21], [24]. One concern we have is regarding the unification of developers contributions, across multiple archives (i.e., mailing list and version-control system), to a single alias. However, core developers have an interest in being recognized for each contribution they make, therefore, maintaining multiple aliases would not be productive. For this reason, we think this issue has limited influence on developer classifications.

Internal Validity: We quantify the agreement between different operationalizations in terms of Cohen's kappa. For these experimental conditions, we required a probabilistic definition of agreement, because a non-error-tolerant agreement metric would be too strict to yield practical results. Cohen's kappa requires some degree of interpretation though, so we have conservatively chosen thresholds that have been established in the literature.

The results of the developer survey depend partially on individual perceptions. To limit this threat, we designed the questionnaire such that multiple developers classified the same developer and we then took the average classification to limit individual bias.

External Validity: The results of our study are based on the analysis of ten open-source projects. Although, the projects do represent a broad spectrum in several dimensions, they are still limited to relatively successful, mature, and large projects. So, the results may not be relevant to immature or very small projects. Likewise, some projects, while having significant commercial involvement (e.g., Linux), are still in the end open source, and it is not yet clear if these results hold for commercial projects.

VIII. Conclusion

Information on developer roles is crucial to understanding the collaborative dynamics of software projects. In particular, accurate knowledge of developer roles provides insight into which developers are appropriate for coordinating other developers' efforts, given their current skill set and expertise. In large, globally-distributed projects, this kind of insight can provide enormous efficiency advantages by reducing the overhead associated with developer coordination [9], [13].

In an empirical study of ten substantial open-source projects, we established evidence that the commonly used count-based operationalizations of developer roles are outperformed by network-based operationalizations in terms of reflecting developer perception. While offering some utility for identifying developer roles, the insights count-based operationalizations can provide are clearly limited, in particular, with regard to the manifold relationships between developers, which may even vary over time. As a novel contribution, we use fine-grained developer networks to establish a relational perspective on developer roles. A key hypothesis is that developer roles should manifest distinctly in the organizational structure, which is also supported by a survey among developers. To this end, we have proposed a number of corresponding network metrics, such as positional stability, hierarchy, and a core–peripheral block model, to explore structural characteristics that capture differences between core and peripheral developers.

Our results suggest that a network perspective can offer valuable insights regarding developer roles that are concealed by non-relational operationalizations. For example, the core group is comprised of the most heavily coordinated developers, and peripheral developers are more likely to coordinate with core developers than with other peripheral developers. The richness of a network perspective has only begun to be explored, and we hope that our work establishes the foundation and inspiration to explore further.

Acknowledgment

We thank all participants of the online survey. This work has been supported by Siemens and the DFG grants AP 206/4, AP 206/5, and AP 206/6.

REFERENCES

[1] G. Bavota, B. Dit, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia. An empirical study on the developers' perception of software coupling. In *Proc. International Conference on Software Engineering*, pages 692–701. IEEE, 2013.

[2] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan. Mining email social networks. In *Proc. International Workshop on Mining Software Repositories*, pages 137–143. ACM, 2006.

[3] C. Bird, A. Gourley, P. Devanbu, A. Swaminathan, and G. Hsu. Open borders? Immigration in open source projects. In *Proc. International Workshop on Mining Software Repositories*. IEEE, 2007.

[4] C. Bird, D. Pattison, R. D'Souza, V. Filkov, and P. Devanbu. Latent social structure in open source projects. In *Proc. International Symposium on Foundations of Software Engineering*, pages 24–35. ACM, 2008.

[5] C. M. Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[6] S. P. Borgatti and M. G. Everett. Models of core/periphery structures. *Social networks*, 21(4):375–395, 2000.

[7] U. Brandes and T. Erlebach. *Network Analysis: Methodological Foundations*. Springer, 2005.

[8] M. Cataldo and J. D. Herbsleb. Communication networks in geographically distributed software development. In *Proc. International Conference on Computer Supported Cooperative Work*, pages 579–588. ACM, 2008.

[9] M. Cataldo and J. D. Herbsleb. Coordination breakdowns and their impact on development productivity and software failures. *IEEE Transactions on Software Engineering*, 39(3):343–360, 2013.

[10] K. Crowston and J. Howison. The social structure of free and open source software development. *First Monday*, 10(2), 2005.

[11] K. Crowston, K. Wei, Q. Li, and J. Howison. Core and periphery in free/libre and open source software team communications. In *Proc. International Conference on System Sciences*, pages 45–56. IEEE, 2006.

[12] C. de Souza, J. Froehlich, and P. Dourish. Seeking the source: Software source code as a social and technical artifact. In *Proc. International Conference on Supporting Group Work*, pages 197–206. ACM, 2005.

[13] C. R. B. de Souza and D. F. Redmiles. The awareness network, to whom should I display my actions? And, whose actions should I monitor? *IEEE Transactions on Software Engineering*, 37(3):325–340, 2011.

[14] T. T. Dinh-Trong and J. M. Bieman. The FreeBSD project: A replication case study of open source development. *IEEE Transactions on Software Engineering*, 31(6):481–494, 2005.

[15] I. Feinerer and W. Mauerer. *tm.plugin.mail: Text Mining E-Mail Plug-In*, 2014. R package version 0.1-1.

[16] J. D. Hamilton. *Time Series Analysis*, volume 2. Princeton University Press, 1994.

[17] J. D. Herbsleb and A. Mockus. Formulation and preliminary test of an empirical theory of coordination in software engineering. In *Proc. European Software Engineering Conference and the International Symposium on Foundations of Software Engineering*, pages 138–137. ACM, 2003.

[18] C. Jensen and W. Scacchi. Role migration and advancement processes in OSSD projects: A comparative case study. In *Proc. International Conference on Software Engineering*, pages 364–374. IEEE, 2007.

[19] C. Jergensen, A. Sarma, and P. Wagstrom. The onion patch: Migration in open source ecosystems. In *Proc. European Software Engineering Conference and the International Symposium on the Foundations of Software Engineering*, pages 70–80. ACM, 2011.

[20] M. Joblin, S. Apel, and W. Mauerer. Evolutionary trends of developer coordination: A network approach. *Empirical Software Engineering*, 2017. Online first.

[21] M. Joblin, W. Mauerer, S. Apel, J. Siegmund, and D. Riehle. From developer networks to verified communities: A fine-grained approach. In *Proc. International Conference on Software Engineering*, pages 563–573. IEEE, 2015.

[22] J. R. Landis and G. G. Koch. The measurement of observer agreement for categorical data. *Biometrics*, 33(1):159–174, 1977.

[23] C. Manteli, B. Van Den Hooff, and H. Van Vliet. The effect of governance on global software development: An empirical research in transactive memory systems. *Information and Software Technology*, 56(10):1309–1321, 2014.

[24] A. Meneely and L. Williams. Socio-technical developer networks: Should we trust our measurements? In *Proc. International Conference on Software Engineering*, pages 281–290. ACM, 2011.

[25] A. Mockus, R. T. Fielding, and J. D. Herbsleb. Two case studies of open source software development: Apache and Mozilla. *ACM Transactions Software Engineering Methodology*, 11(3):309–346, 2002.

[26] K. Nakakoji, Y. Yamamoto, Y. Nishinaka, K. Kishida, and Y. Ye. Evolution patterns of open-source software systems and communities. In *Proc. International Workshop on Principles of Software Evolution*, pages 76–85. ACM, 2002.

[27] G. A. Oliva, F. W. Santana, K. C. M. de Oliveira, C. R. B. de Souza, and M. A. Gerosa. Characterizing key developers: A case study with Apache Ant. In *Proc. International Conference on Collaboration and Technology*, pages 97–112. Springer, 2012.

[28] E. Ravasz and A.-L. Barabási. Hierarchical organization in complex networks. *Physical Review E*, 67(2), 2003.

[29] G. Robles and J. M. Gonzalez-Barahona. Contributor turnover in libre software projects. In *Open Source Systems*, pages 273–286. Springer, 2006.

[30] G. Robles, J. M. Gonzalez-Barahona, and I. Herraiz. Evolution of the core team of developers in libre software projects. In *Proc. Mining Software Repositories*, pages 167–170. IEEE, 2009.

[31] F. Shull, J. Singer, and D. I. Sjøberg. *Guide to Advanced Empirical Software Engineering*. Springer, 2007.

[32] A. Terceiro, L. R. Rios, and C. Chavez. An empirical study on the structural complexity introduced by core and peripheral developers in free software projects. In *Proc. Brazilian Symposium on Software Engineering*, pages 21–29. IEEE, 2010.

[33] Y. Ye and K. Kishida. Toward an understanding of the motivation open source software developers. In *Proc. International Conference on Software Engineering*, pages 419–429. IEEE, 2003.

[34] X. Zhang, T. Martin, and M. E. J. Newman. Identification of core-periphery structure in networks. *Physical Review E*, 91, 2015.