The Patch-Flow Method for Measuring Inner Source Collaboration

Maximilian Capraro
Friedrich-Alexander-Universität Erlangen-Nürnberg
Erlangen, Germany
maximilian.capraro@fau.de

Michael Dorner*
Friedrich-Alexander-Universität Erlangen-Nürnberg
Erlangen, Germany
michael.dorner@fau.de

Dirk Riehle
Friedrich-Alexander-Universität Erlangen-Nürnberg
Erlangen, Germany
dirk.riehle@fau.de

ABSTRACT

Inner source (IS) is the use of open source software development (SD) practices and the establishment of an open source-like culture within an organization. IS enables and requires developers to collaborate more than traditional SD methods such as plan-driven or agile development. To better understand IS, researchers and practitioners need to measure IS collaboration. However, there is no method yet for doing so. In this paper, we present a method for measuring IS collaboration by measuring the patch-flow within an organization. Patch-flow is the flow of code contributions across organizational boundaries such as project, organizational unit, or profit center boundaries. We evaluate our patch-flow measurement method using case study research with a software developing multi-industry company. By applying the method in the case organization, we evaluate its relevance and viability and discuss its usefulness. We found that about half (47.9%) of all code contributions constitute patch-flow between organizational units, almost all (42.2%) being between organizational units working on different products. Such significant patch-flow indicates high relevance of the patch-flow phenomenon and hence the method presented in this paper. Our patch-flow measurement method is the first of its kind to measure and quantify IS collaboration. It can serve as a base for further quantitative analyses of IS collaboration.

CCS CONCEPTS

• **General and reference → Empirical studies**; **Measurement**; Metrics; General conference proceedings; • **Software and its engineering → Open source model**; *Software creation and management*; *Software development process management*;

KEYWORDS

Inner source, internal open source, inner source measurement, patch-flow, open source, open collaboration, software development collaboration measurement, inner source metrics

*Second and third author contributed to this paper with similar significance. Their sequence is determined by alphabetical order.

MSR '18, May 28–29, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5716-6/18/05…$15.00
https://doi.org/10.1145/3196398.3196417

ACM Reference Format:
Maximilian Capraro, Michael Dorner, and Dirk Riehle. 2018. The Patch-Flow Method for Measuring Inner Source Collaboration. In *MSR '18: MSR '18: 15th International Conference on Mining Software Repositories , May 28–29, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3196398.3196417

1 INTRODUCTION

Open source software plays a key role in today's software industry. Open source (OS) development tools help to build software and open source components are used as part of proprietary software products. OS is recognized to be capable of delivering high quality software [5]. The software industry has shown a significant interest in benefiting not only from OS's outcomes (the software components and tools) but also from the adoption of software development (SD) practices exercised in the OS world [30].

The "use of open source [SD] practices and the establishment of an open source-like culture within organizations" is called inner source (IS) [2]. While IS SD is similar to and shares attributes with OS SD, IS opens up the development only internally within the environment of one organization [7]. In addition to the interest of the software industry, the research community has shown interest in IS as a research topic indicated by a steady stream of scientific publications [2].

However, the majority of scientific literature regarding IS presents only qualitative results such as case study reports or taxonomies of IS programs and projects. There is not yet a quantitative study discussing the extent of IS and participating parties in it and no method to measure IS collaboration within an organization. We believe that such a method is needed by both researchers and practitioners. Researchers can use it as a base measurement and foundation for sophisticated quantitative models such as evaluation models for IS programs or metrics for IS project performance. Practitioners can use it to derive an overview of the participants in and the state of their IS program as well as define key performance indicators based on the methods output.

In this paper, we present a method to quantify IS collaboration by measuring an organization's patch-flow. In OS, a patch is a code contribution from an individual external to an OS project. The word patch is historically derived from the "patch files" (produced and consumed by the "diff" and "patch" commands) that some OS projects use in their contribution workflow. Today, patches can have different forms (for example a pull request on a software forge like GitHub). Typically, the contributor of a patch does not have write privileges to a project's code base. The patch has to be picked up and integrated by a person with those privileges (called

a committer). In this way, OS projects perform quality assurance of contributed code. Patch-flow then is the flow of such patches across organizational boundaries. In OS, boundaries might be between contributing organizations and an OS project. In IS, such boundaries are intra-organizational boundaries like organizational unit, project, or profit center boundaries. When measuring the patch-flow, it is not sufficient to simply count patches over time. One must address the organizational structure contextual to the patch.

Our paper answers the following research question:

- How to measure IS collaboration within a software developing organization?

We answer this research question by presenting a method to measure patch-flow. In detail, this paper contributes the following:

- Definition of the patch-flow phenomenon
- A patch-flow measurement method for measuring IS collaboration
- An evaluation of the patch-flow measurement method using case study research with a software developing multi-industry company

The remainder of this paper is structured as follows. Section 2 gives an overview of the related work detailing prior work regarding IS and measuring SD collaboration. Section 3 introduces the patch-flow measurement method by defining the patch-flow phenomenon, data structures to represent patch-flow and process to measure it. Section 4 describes our evaluation approach for the patch-flow measurement method using case study research. Section 5 reports the evaluating case study where we apply the method. Section 6 discusses the findings from the case study and proposes future work. Section 7 closes the paper with a conclusion.

2 RELATED WORK

We first discuss prior work on IS to give an introduction to the field. Then we lay out how related work has measured IS collaboration and SD collaboration in general.

2.1 Inner Source Definition

The term IS has been coined by O'Reilly [24]. IS is the "use of open source software development practices and the establishment of an open source-like culture within organizations" [2].

In IS, selected software components are made available as IS projects. An IS project "is a software project with the goal to develop and maintain inner source software. [...] IS projects are [like] open source projects that do not have a defined end date. As in open source, the name of the project is often also used to address the ISS component" [2]. Inner source software (ISS) is the "software product [or component] that is developed within an IS context" [31].

Developers within the organization can read the source code of an IS project and use it as part of their work. In addition to reading the source code, developers can contribute patches to the IS project. A patch is a code contribution made by a developer who is external to an IS project. A developer is considered external to a project if not a member of the organizational unit owning the IS project. Typically, patches need approval by the committers of an IS project who decide whether to reject a patch or enact it by integrating it into code base [12, 27]. A committer is an individual with write ("commit") privileges to a project's code base. An IS project can have one or many committers.

IS projects communicate openly: Every individual within the company can read and participate in discussions [22]. Open communication should not only be public within the organization but also archived, written, and complete [26]. A common tool for exercising open communication is a mailing list [34].

2.2 Measuring Inner Source Collaboration

Researchers and practitioners have published a steady stream of case studies with companies that have adopted or are aiming to adopt IS including DTE Energy [29], Ericsson [32], Hewlett-Packard [7, 20], IBM [36], Kitware [19], Lucent [12, 13], Nokia [17], Paypal [23], Philips [35], Rolls-Royce [30], and SAP [27].

These studies provide qualitative discussions or anecdotal evidence of the IS phenomenon. To the best of our knowledge, no study delivers an in-depth quantitative analysis of IS collaboration or method to measure it.

However, some studies report simple counting metrics. Organizations quantify the size of an IS program by counting the number of IS projects in the portfolio [7, 27]. Other organizations extend these metrics by counting the number of developers contributing to or being committers in each IS project [17, 32]. In contrast to our work, they do not discuss how actively IS projects receive IS contributions. Gurbani et al. [12] measured the code churn in their IS project. In contrast to our work, they do not measure what parties are involved in contributing to the IS project.

Vitharana et al. [36] argue that measurements and metrics regarding IS need to account for different parties involved in IS collaboration. Our patch-flow measurement method does exactly this by outputting data that shows which parties receive how much IS code contributions from which parties within the organization.

2.3 Measuring Collaboration

Prior research addresses measurement of software development collaboration in proprietary and open source development.

2.3.1 Social Network Analysis. A variety of studies analyze software development collaboration using social network analysis (SNA). Over the last years, "applying SNA to software development teams has been a heavily researched topic" [21]. Previous research utilizes primarily two types of social networks [21]: Developers networks [3, 4, 15, 18, 28, 33] and contribution networks [25].

Developer networks differ significantly from the patch-flow graphs we present. A developer network is a graph showing relationships (edges) between developers (nodes). Researchers construct developer networks by identifying "records of social or technical connections" between developers from source code management systems, communication logs, issue tracking systems, or other sources [21]. Developer networks are capable of representing relationships among developers [21] and model the structure of development communities [15]. In contrast, our paper presents patch-flow graphs that use the structure of a given organization and model to what extent parties within this structure (typically organizational units) contribute to one another.

Contribution networks are graphs showing contributions of developers to one or many source code files [21, 25]. A contribution network shows source code files and developers as nodes; contributions as directed edges from a developer to a file. In both contribution network and patch-flow graph, directed edges represent code contributions. However, the nodes differ in semantics and granularity. In a patch-flow graph, contributing nodes do not represent individual developers but parties within an organization. Receiving nodes do not represent low-level source code files, but parties (organizational units or IS projects) receiving contributions. Nodes in the patch-flow graph can represent parties at different levels in the organizational hierarchy. Thus, the patch-flow graph can be seen as a generalization of contribution networks. In contrast to the contribution network, a patch-flow graph shows aggregated contributions using information on the organizational structure. Consequently, collaboration can be made visible even for large organizations without the information overload a contribution network would suffer from.

2.3.2 Other Approaches. Gousios et al. [9] and Kalliamvakou et al. [16] present a taxonomy to measure and classify contributions to open source software projects. Similar to our work, the contribution is the key element of their model. Their method focuses on qualifying the contributions of an individual towards an open source project. In contrast to our method, their method disregards organizational structures because it does not aim to measure how contributions flow across an organization.

Begel et al. [1] present a framework for the extraction of development meta data and applies it at Microsoft. While they do not aim to measure the patch-flow, the described data structures are sufficient to construct a patch-flow graph.

3 PATCH-FLOW MEASUREMENT METHOD

Based on our experience from prior research on IS, we designed a method for measuring an organization's patch-flow and with that IS collaboration.

3.1 Patch-Flow Phenomenon

We define the following concepts:

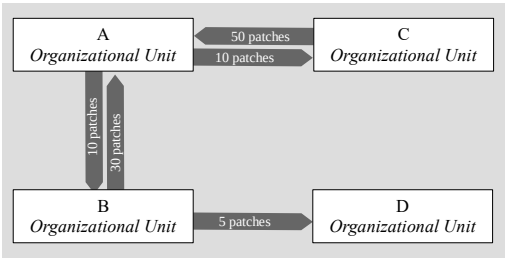
- A code contribution is any code change performed on a software component.
- A patch is a code contribution made by a developer who is external to a project.
- Patch-flow is the flow of patches across organizational boundaries such as project or organizational unit boundaries within a company.

A developer is considered external to a project if not a member of the organizational unit owning the IS project. Typically, patches need approval by the committers of an IS project who decide whether to reject a patch or enact it by integrating it into code base [12, 27].

3.1.1 Example. Figure 1 displays example patch-flow involving four organizational units A, B, C, D. The white boxes are organizational units; the gray arrows indicate patch-flow between two organizational units.

In the example, ten patches flow from organizational unit A to B. That means developers allocated to work for organizational unit A contributed ten patches to components that are owned by organizational unit B. A total amount of 80 patches is received by A that only contributed 20 patches to B and C in total.

Figure 1: Example patch-flow between four organizational units



3.1.2 Relationship to Inner Source. An individual within an organization takes part in IS collaboration by contributing to an IS project provided by another organizational unit (across an organizational boundary). Contributions to an IS project can have multiple shapes: In addition to a code contribution, an individual may contribute by reporting a bug, reviewing the contribution of somebody else, taking part in a mailing list discussion or by other means. In this paper, we do not consider the flow of non-code contributions but only the flow of code contributions. Because software code is the primary artifact resulting from a software development effort and a majority of other contributions will eventually result in code contributions, the patch-flow is an appropriate measure for IS collaboration. More IS collaboration equals more patch-flow.

3.2 Data Structures

We present an object-oriented model for representing the patch-flow within an organization. From objects adhering to this model, patch-flow graphs in differing granularity can be constructed.

3.2.1 Patch-Flow Model. Figure 2 displays a simplified object-oriented model in UML2 annotation capable of representing patch-flow data. The key element is the code contribution (CodeContribution class) with attributes indicating what change was performed on which files and when. For each code contribution, it contains the person (Person class) authoring a code contribution and the IS projects (InnerSourceProject class) receiving it. Each person and IS project is associated with an organizational unit (OrgUnit class). Organizational units are modeled using the composite design pattern [8]: An organizational unit can be composed of child organizational units.

3.2.2 Patch-Flow Graph. We introduce the term patch-flow graph as follows:

- A patch-flow graph is a directed weighted graph with organizational units as nodes and weighted edges representing patch-flow between these nodes.

Each edge weight represents the number of patches flowing. From the data measured in an organization (instantiating the classes presented in the previous section), multiple patch-flow graphs can be constructed depending on the granularity of organizational units.

Figure 2: Simplified patch-flow flow data model as UML2 class diagram

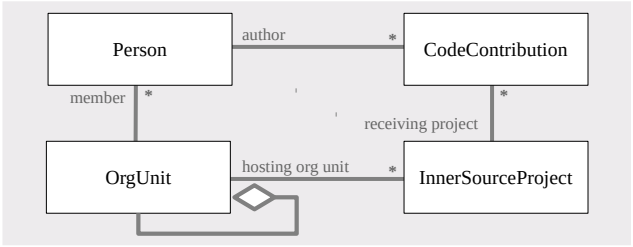
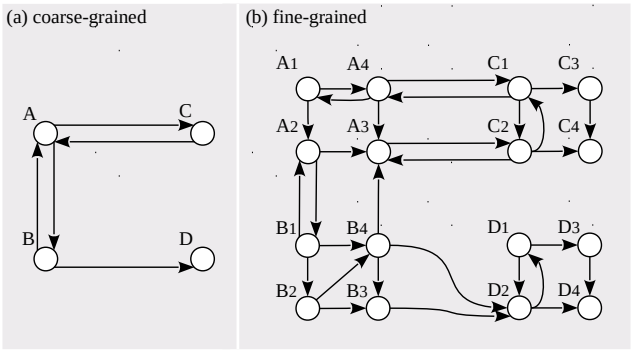


Figure 3: Patch-flow graphs with different granularity



To distinguish patch-flow between organizational units of different granularity, we apply the concept of levels to the organizational hierarchy. In a tree, the level of each node is $n + 1$ with n being the level of its parent. The root node always has the level 0. Translated to organizational hierarchies, the organization itself has level 0 and its top-level organizational units have level 1. Their children organizational units have level 2 etc. Level 0 is considered the highest level. We define the following terms:

- The lowest common ancestor (LCA) of two organizational units is the lowest node that has both as descendants.
- Patch-flow crosses level n if and only if $n < n_{lca}$ with n_{lca} being the level of the lowest common ancestor of the contributing and the receiving organizational unit.

Code contributions between descendants and ancestors not considered patch-flow. Patch-flow crossing level n always crosses level $n+1$. The *highest level crossed* of a patch-flow ($n = n_{lca} - 1$) serves as a metric for the distance between the two involved organizational units: Two coarse-grained business units in a large conglomerate have a higher distance, then two teams within one of these business units.

Figure 3 shows two patch-flow graphs visualizing patch-flow in the example organization from the previous section. The organization is composed of organizational units A, B, C, D (level 1). They each have four children numbered with $A_1, A_2, \ldots$ (level 2).

- Part (a) shows the patch-flow crossing level 1 (between four organizational units A, B, C, D).
- Part (b) is more fine-grained showing the patch-flow crossing level 2 (between the children of A, B, C, D).

Constructing different patch-flow graphs allows researchers and practitioners to study the patch-flow between organizational units on different levels in the organizational hierarchy. While this "drilling down" increases the level of detail, in organizations with a lot of patch-flow this might lead to information overload and reduce the comprehensibility of the graph for a human reader.

3.3 Measurement Process

Before measuring the patch-flow in an organization, we assume that a researcher or practitioner defined a scope by deciding which IS projects to include in measurement. There are valid reasons for a narrow scope and for not including all software developed within the organization: For example, a complete list of all IS projects and software components is not always available or an organization could decide to allow patch-flow measurement only for selected IS projects due to the protection of intellectual property or other sensitive data. IS collaboration can occur regarding software that is not formally part of an IS project. Thus, projects with allegedly no IS collaboration can be considered as well.

The patch-flow in a given organization can be measured by executing the following activities:

(1) *Extraction of code contributions.* Extract code contributions regarding the IS projects in the scope. Typically, code contributions result in commits to a source code repository and can be extracted from there.
(2) *Mapping of code contributions to IS projects.* Map the receiving IS project to each code contribution.
(3) *Extraction of organizational data.* Extract data about the structure of the studied organization. Organizational data can be extracted from a variety of databases like directory services or project management tools. However, the complexity of organizational modeling might require manual extraction or cleaning of the data.
(4) *Identification of author.* Identify the author of each code contribution. Depending on the source code repository used, authors might only be identified by pseudonym strings or other identifiers like email addresses.
(5) *Mapping of authors to organizational unit.* Map the authors of a code contribution to their organizational unit.
(6) *Mapping of IS projects to organizational units.* Map the IS projects to the organizational units responsible for them. For some IS projects, it might not be possible to allocate an organizational unit.

The activities do not need to be executed in the given order. For example one could decide to identify authors (activity 4) and subsequently extract organizational data (activity 3) only for active authors' and IS projects'.

When measuring the patch-flow incrementally, the measurement costs are reduced significantly after the first increment because a large number organizational units and authors are already identified. Ideally, all activities are automated to reduce measurement costs and risk of human errors during measurement.

4 EVALUATION APPROACH: CASE STUDY RESEARCH

We evaluated the patch-flow measurement method by applying it in an industry case study following Yin [37]. We developed a case study protocol [37] to outline our case study design including data gathering and evaluation mechanisms in detail. We evaluate three dimensions of our patch-flow measurement method:

(1) We evaluate the *relevance* of the patch-flow phenomenon and measurement method in the context of the case study's findings.
(2) We evaluate the method's *viability* within the context of the case study organization.
(3) We demonstrate the *usefulness* of the patch-flow data and patch-flow graphs to represent IS collaboration within an organization by analyzing the measured patch-flow and discussing it in-depth in the context of the case organization.

4.1 Case Selection

We searched for a software developing organization with established development processes, a large number of developers, and existing IS collaboration. From our professional network, we identified an organization fulfilling these requirements. Upon request of our partners in the case organization, we do not disclose the real name of the organization.

The studied organization is a multi-industry company with significantly more than 10.000 developers. It operates internationally but the majority of development work is performed within one country. Most dominantly, it uses (traditional) plan-driven development processes.

The company is structured into multiple segments. Within one segment, we identified 18 test infrastructure components that are set up as IS projects. The source code of these components is open for all developers within the organizations to read and contribute to. We measure the patch-flow regarding these 18 IS projects.

4.2 Data Gathering

We employed two data gathering mechanisms: We applied the patch-flow measurement method for the 18 identified test infrastructure IS projects. In parallel, we gathered qualitative data to broaden our understanding of the case organization and interpret the measured patch-flow data.

4.2.1 Iterative Gathering of Qualitative Insights. We performed three unstructured interviews with the engineering manager responsible for the test infrastructure development. Towards the end of our case study inquiry, we performed an extensive workshop with more than 10 employees in development and project management roles within the studied organization. Table 1 presents details about the interviews and workshop. We reference each interview and workshop throughout the paper using its ID.

In interviews I1 and I2, we inquired about what how to measure and interpret base data for the patch-flow measurement. In interview I3 and workshop W1, we presented different visualizations of the patch-flow data, asked for the employees' interpretations, presented our interpretations, and collected feedback regarding our interpretations. While the focus of this paper is on quantifying IS collaboration, we used this qualitative feedback to broaden our understanding of the collaboration and context within the studied organization.

Table 1: Gathered qualitative data

ID	Type	Participants	Topics
I1	Interview	Eng. manager test infrastructure	IS collaboration, organizational structure
I2	Interview	Eng. manager test infrastructure	Organizational structure
I3	Interview	Eng. manager test infrastructure	Interpretation of visualizations, feedback
W1	Workshop	Employees in multiple roles	Interpretation of visualizations, feedback

4.2.2 Application of Patch-Flow Measurement Method. We followed the patch-flow measurement method discussed in the previous section.

The studied organization stores the source code of the 18 identified IS projects in a Microsoft Team Foundation Server (TFS) code repository. With the help of an on-site engineer, we utilized a TFS export script proprietary to the studied organization to extract the code contributions from the repository (activity 1 from section 3.3). We considered code contributions between April 1st, 2015 and June 30th, 2016 (boundaries included). Code for each IS project is located in a designated sub-directory of the repository. We utilized the directory paths to determine for each code contribution the receiving IS project (activity 2).

Organizational data sources like the organization's LDAP directory did not provide a detailed description of the organizational structure. While they were containing the high-level organizational units of each developer (i.e. business units), they did not contain the lower level organizational units (i.e. project teams). An engineering manager responsible for the test infrastructure development manually assembled information on the organizational structure into a machine readable format (activity 3). The studied organization is a matrix organization. Employees report to their disciplinary superior (disciplinary organization) and at the same time are allocated to a project team (project organization). We use the project organization for patch-flow measurement because it represents everyday work routine. We consulted with employees of the organization who expressed that the disciplinary organization had little impact on their everyday work routine but merely represented where and how an individual was hired (interview I1, I2). In addition, software artifacts are owned by organizational units of the project organization.

Due to privacy concerns, we did not get access to the employee database. An employee of the studied organization manually searched the identifiers of each code contribution author (activity 4), mapped them to their organizational unit (activity 5), and using an internal database assigned the owning organizational unit (activity 6). Like Guzzi et al. [14], we observed that the employee database contained no historical data. As a consequence, we could not identify the author of every code contribution.

We excluded code contributions from analysis where we could not identify the author of the change (116 code contributions), the organizational unit of the author (14 code contributions), or an IS project receiving the contribution (827 code contributions). In addition, we considered only code contributions with actual changes to the code: We excluded changes induced by repository management tasks like branching or merging. The data gathering resulted in 2194 not-excluded code contributions.

5 EVALUATION RESULTS: PATCH-FLOW AT STUDIED ORGANIZATION

We found significant patch-flow between the organizational units of the studied organization.

5.1 Organizational Structure

Figure 4 displays an excerpt of the organizational structure of the studied organization. The nodes (circles) represent organizational units. The edges (lines) indicate child-parent relationships between organizational units. The nodes are annotated with letters that are used to construct organizational unit identifiers (if a node is annotated with *b* and its parent node with *a*, then the identifier is *ab*). The excerpt only contains organizational units that took part in IS collaboration by contributing patches to or providing an IS project.

The organizational hierarchy is at maximum six levels deep. The studied organization has a basic model of the structure defining a type for organizational units on each level (interview I2). Using this type information, we adjusted the level for *ab*, *baba*, and their descendants. We assigned each organizational level one color that we use to identify the level throughout this paper.

We refer to the most coarse-grained organizational units as segments (level 1). In the studied organization, segments are essentially companies within the company. They offer independent services and product portfolios, and cater to different markets (interview I1, I2). The segments have more than 10,000 employees each.

Segments are composed of business units (level 2). A business unit encapsulates one specific product domain within the market of its segment. Organizational units on level 5 each develop a different product or a small set of tightly related products. Teams (level 6) are the most fine-grained organizational units and typically consist of 5 to 15 developers. Each team has a specific technical task regarding one part of a product. Figure 4 displays each team with its full identifier consisting of the parent organizational unit (e.g. *aaaaa*) and a team identifier (t_i, with index i of $a, b, c, \ldots$).

Eleven teams own one or more IS projects (gray boxes in figure 4). All IS projects are provided by teams in segment *a*. Each level 4 descendant of segment *a* provides an IS project.

In the studied organization, all IS projects and developers belong to a team (leaf node). This is not necessarily the case in every organization. For example in other organizations higher level organizational units could own an IS project or employees assigned to a higher level organizational unit could decide to contribute patches.

5.2 Patch-Flow Overview

Figure 5 shows the patch-flow aggregated by level 4 organizational units as a graph. For comprehensibility, we use a different notation than for the previous patch-flow graphs: The white boxes represent level 4 organizational units. The figure also displays a part of the organizational hierarchy. Level 4 organizational units are contained by gray boxes representing level 3 organizational units. Gray lines around these boxes indicate level 2 organizational units (business units). Directed edges between the level 4 organizational units represent the patch-flow.

The width of an edge shows its weight (the amount of patches flowing). The edges spin mathematically positive (counter clockwise). Both, the color and line-type of an edge indicates the highest level crossed by the patch-flow. In the studied organization, all patch-flow crossing level 2 also crosses level 1. Consequently, the figure only contains the colors for patch-flow with highest level crossed equals 1, 3, and 4.

We measured significant patch-flow regarding the sampled IS projects. In total 820 code contributions (37.4% of all contributions to the IS projects) constituted patch-flow across a level 4 boundary or higher.

Seven level 4 organizational units contribute or receive patches crossing level 4 and are consequently included in figure 5. Three of them (*aaad*, *aaca*, *baba*) do not receive patches. The organizational units *aaad* and *aaca* receive no patches crossing level 4 despite hosting IS projects. The organizational unit *baba* receives no patches as a consequence of our sampling: We considered only IS projects of segment *a*. Consequently, we do not find any patch-flow into organizational units of other segments.

Identifying contribution activity. How much an organizational unit's developers contribute to IS projects of other organizational units varies. On the one hand, developers of the organizational unit *aba* contribute no patches to other organizational units (indicated by no edge). Developers of other organizational units (for example *aaca*) contribute only a few patches. On the other hand, developers of select organizational units contribute a significant number of patches to IS projects in other organizational units (*aaad* contributing 315 patches, *aaba* contributing 172 patches). The patch-flow graph is capable of expressing how much an organizational unit contributes to other organizational units' IS projects (sum of the weight of all outgoing edges).

Identifying contribution receival. Organizational units receive a varying number of outside patches. Two organizational units do not receive patch-flow despite hosting an IS project within our sample (*aaca*, *aaad*). The organizational unit *aaaa* receives the highest number of patches (606). The patch-flow graph is capable of expressing how many patches an organizational unit receives from other organizational units (sum of the weight of all incoming edges).

Identifying collaboration relationships. An organizational unit might collaborate only with select organizational units. For example, we observed intense collaboration between the children organizational units of *aaa* and *aab* compared to other level three organizational units. Children of *aaa* contribute 131 patches to *aab*; children of *aab* contribute 172 patches to *aaa*. The patch-flow graph is capable of expressing how intense two organizational units collaborate with one another (sum of edge weights between these two organizational units).

Figure 4: Organizational structure including all organizational units participating in IS collaboration

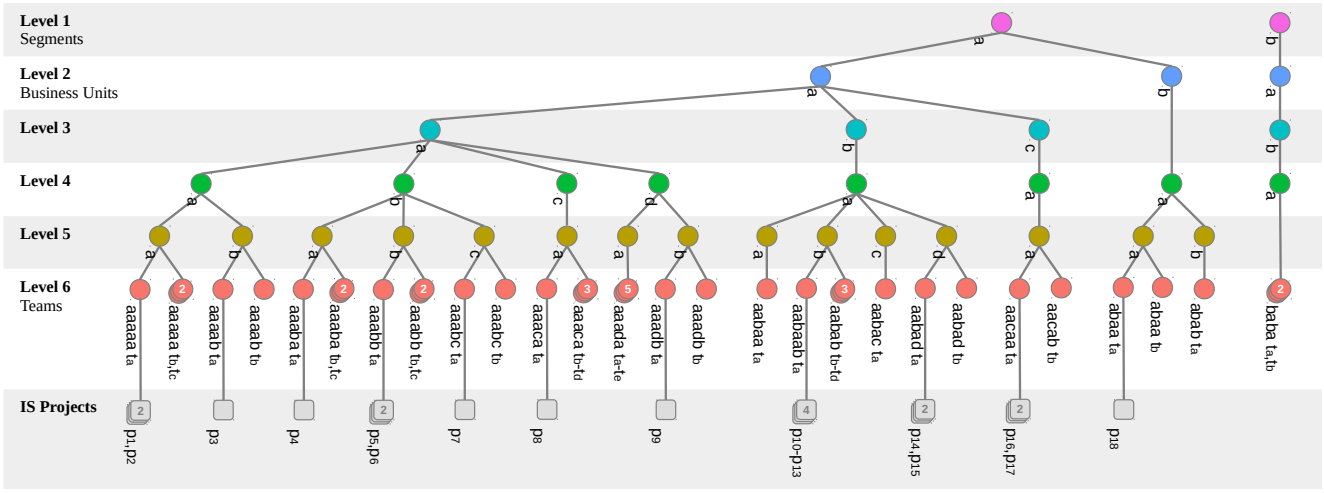
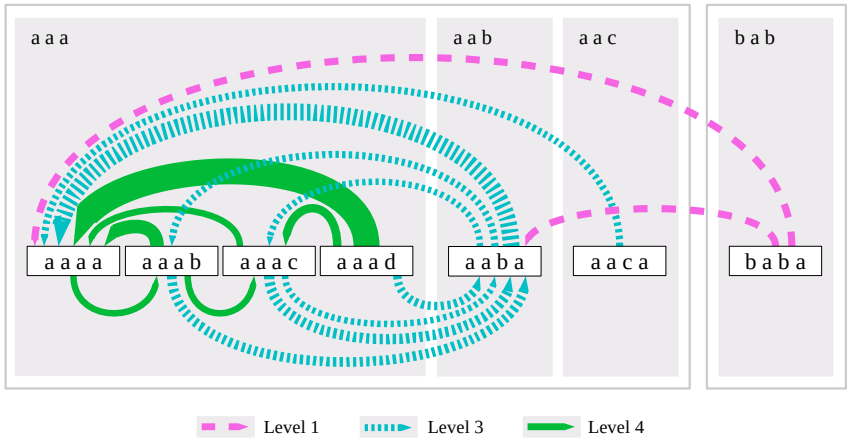


Figure 5: Patch-flow graph showing patch-flow between the level 4 organizational units



5.3 Patch-Flow Over Time

In the previous section, we presented a patch-flow graph that was augmented with hierarchical information regarding the organization. With an increasing amount of involved organizational units or hierarchical depth of the organization, such a graph can quickly become incomprehensible. However, hierarchical information must not be neglected: A patch-flow between two low-level organizational units (e.g. teams) can have a different meaning than a patch-flow between two top level organizational units (e.g. segments): For example, there might be more and harder challenges to overcome establishing IS collaboration among large segments than among teams within the same segment.

Figure 6 provides an alternative representation of the patch-flow considering the organizational hierarchy. The x-axis presents the time. The y-axis displays the patch-flow relative to the total amount of code contributions to the IS projects per month. The color and line type of each line indicates the highest level crossed by the patches. For example, the red line shows the percentage of patches flowing between level 6 organizational units (teams). The yellow line shows the percentage of patches flowing between level 5 organizational units.

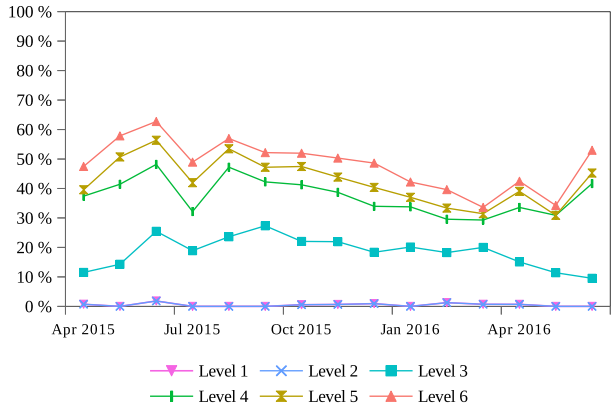
We observe intense IS collaboration (significant patch-flow) across level 6 to level 4 of the organization:

- 47.9% of all code contributions to the IS projects in our measurement period flow across level 6 (teams)
- 42.4% across level 5
- 37.4% across level 4

There is less patch-flow among level 3 organizational units:

- 18.5% across level 3

Figure 6: Patch-flow relative to all code contributions over time by highest level crossed



We observe nearly absent collaboration (insignificant patch-flow) among level 2 and 1 organizational units:

- 0.4% across level 2 (business units)
- 0.4% across level 1 (segments)

Patch-flow across level 6 to level 4 of the organization is routine with the sampled IS projects. More than half of the contributions to IS projects come from other teams; over a third from other level 4 organizational units. There is less patch-flow crossing lower level boundaries. Business units and segments do not significantly collaborate with one another. Despite the openness brought by inner source practices they can still be considered so called "silos" (organizational units that do not collaborate).

Identifying silo boundaries. A visualization of the patch-flow crossing different organizational levels as in figure 6 is capable to express on what organizational levels silos have formed and across which levels how much IS collaboration happens.

5.4 Patch-Flow into IS Projects

Figure 7 shows the total number of code contributions received by an IS project. The y-axis lists the IS projects. Each bar along the y-axis indicates the number of received code contributions. The color and pattern of the stacked bars indicate the highest level crossed by the patches. The black striped bar indicates contributions that do not constitute patch-flow (contributions by the team running the IS project).

The IS projects receive a varying number of code contributions. Two IS projects received no code contributions in our measurement period (p9, p18). Seven projects receive code contributions but less than twenty (p2, p6, p10, p12, p13, p15, p16). We consider these IS projects inactive. Four projects (p1, p3, p8, p11) receive more than 200 code contributions.

Figure 8 displays only the projects we consider active. The y-axis is ordered by total amount of received code contributions. In contrast to figure 7, the bars along the x-axis show the percentage of received patches.

Figure 7: Absolute number of code contributions received by IS project

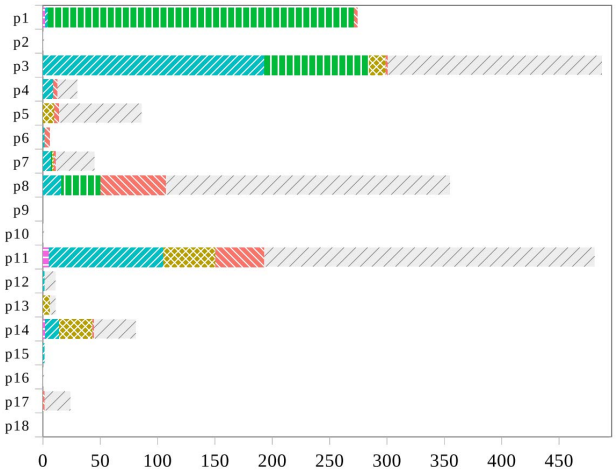
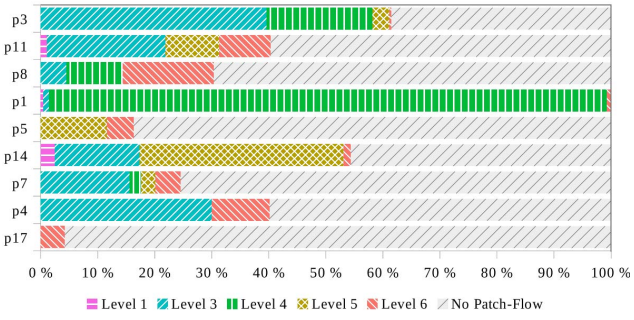


Figure 8: Ratio of patches to all code contributions by IS project



All active IS projects receive patch-flow. However, they receive a varying portion of patch-flow. Project p17 receives the smallest ratio of outside patches (4.2%) and project p3 the largest (61.5%). The IS projects receive patch-flow crossing different levels. While p17 received only patches crossing level 6 (other team), the majority of teams receives a mix of contributions crossing level 6 to 3. Only two projects (p11, p14) receive patches that crossed a level 1 (other segment).

Indicating different reach of IS projects. The measured patch-flow indicates that IS projects acquire different reach within the organization with some receiving patches only from neighboring teams and others attracting contributions even from other segments.

We consider project p1 an outlier because it received 100% patch-flow. The team responsible for the IS project did not perform any code contributions. In total, eleven level 6 organizational units (teams) from six level 4 organizational units contribute to p1. Most patches (90,1%) are contributed by the team $aaadbt_a$. Our contacts at the studied organization confirmed to us that another team

($aaaaat_a$) is the owner of p1 and and coordinates work on this IS project (interview I3, workshop W1). However, $aaadbt_a$ performs a majority of the development and maintenance tasks.

Indicating problems with ownership and component granularity. The significant patch-flow into project p1 (and the tight collaboration it expresses) is not necessarily a positive indicator regarding the studied organization's development setup. We believe outliers receiving higher patch-flow can indicate a problem. In the case of project p1 responsible teams are not doing the actual work. We suggest practitioners facing such a situation to reconsider who are the owners of the IS project. We believe that inconveniently cut or too coarse grained components could lead to a high patch-flow.

6 DISCUSSION

In this section, we discuss the findings from our case study. We first discuss the evaluation of our patch-flow measurement method. Subsequently, we discuss how the patch-flow can serve as an operational definition for IS collaboration, evaluate the trustworthiness of our research and lay out suggested future research.

6.1 Evaluation

We evaluate our patch-flow measurement method based on its relevance, viability, and usefulness.

6.1.1 Relevance. In the evaluating case study, we found that about half (47.9%) of all code contributions constitute patch-flow between organizational units, almost all (42.2%) being between organizational units working on different products. The significant patch-flow measured in the studied organization indicates high relevance of the patch-flow phenomenon and hence the method presented in this paper.

6.1.2 Viability. To evaluate its viability, we applied the patch-flow measurement method in an industry case study with a software developing multi-industry company. Following the measurement activities, we populated the discussed patch-flow data structures. Patch-flow measurement using our method is feasible; the method was applicable to the case study organization.

However, we observed pitfalls practitioners and researchers should be aware of when measuring the patch-flow:

First, the organizational databases stored only incomplete records of the organizational structure. We had to manually collect data regarding the organizational structure. We recommend individuals applying the patch-flow measurement method, to carefully vet systems that might contain data on the organizational structure and to rely on costly manual collection of data only as a last resort.

Second, every IS project at the studied organization has a dedicated team that is responsible for it. Which team is determined as owner of an IS project is crucial for the measured patch-flow. Individuals faced with a project receiving only outside contributions (like p1 in section 5.4) need to carefully re-evaluate who owns this project. Capraro and Riehle [2] discuss that with evolving IS initiatives, some IS projects might be owned by many or even all teams of an organization. In such cases, researchers need to find operational definitions of IS project ownership that fit the context of their study. For example IS projects owned by more than one team can be modeled as belonging to the owners' lowest common ancestor organizational unit.

Third, developers and IS projects are not necessarily assigned to leaf nodes in the organizational hierarchy. Where this is not the case, one should carefully refer to the definitions given in section 3.2.2: A code contribution from an organizational unit to its descendant or ancestor is not patch-flow (because its is not possible that both organizational units have a level $n < n_{lca}$ with n_{lca} being the level of their lowest common ancestor).

6.1.3 Usefulness. Three observations from the case study are relevant to evaluate the usefulness of patch-flow method and its outcomes for practitioners:

First, the case organization invested own resources to support our patch-flow measurement (see section 4.2). This indicates to us, that our contact persons in the organization, expected the results to be useful to them even before measurement started. Second, during workshop W1, the participants used the patch-flow as a base for discussion of their collaboration practices. Participants saw the amount of outside contributions to IS projects and the diversity of contributing teams, as an argument to strengthen the role of committers and staff their active IS projects with more than one committer. That our visualizations led to the discussion of concrete actions, indicates to us, that the they are useful to practitioners. Third, During interview I3 and workshop W1, individual participants inquired about specific additional aggregations of the patch-flow data. This indicates to us, that patch-flow visualizations can deliver an overview of IS collaboration, but that practitioners can use them as a starting point to define additional metrics and visualizations (using the patch-flow data) regarding specific collaboration goals or information needs of their organization.

We believe that the insights delivered by the visualizations in section 5 are useful to both practitioners and researchers: The patch-flow graph enriched with additional hierarchical information, allows to identify hot spots with tight IS collaboration, and organizational units not participating in IS collaboration. In addition to the patch-flow graph, patch-flow data can show what organizational levels are typically crossed by patches, which IS project is developed how actively, and from where each IS project attracts patches.

6.2 Operational Inner Source Definition

On the one hand, some organizations run IS projects without calling them IS projects or even without people being aware that they are performing IS collaboration. For researchers, this can lead to a struggle in establishing construct validity regarding what they claim to be (or not be) IS projects or instances of IS collaboration or IS in general. On the other hand, some organizations might use the term IS project despite a component receiving no or insignificant contributions (like the inactive IS projects in section 5.4). We believe that the patch-flow can be used to establish an operational definition of IS collaboration: IS collaboration is collaboration across organizational boundaries such as project or organizational unit boundaries within a company. Consequently, where there is a flow of patches (or other contributions), there is IS collaboration.

6.3 Trustworthiness

We evaluate the trustworthiness of our results using the quality criteria credibility, dependability, confirmability and transferability [11]. These quality criteria for naturalistic research can be applied to evaluate case study inquiries in general [11], case studies in the context of software engineering [6], and have been applied to discuss the trustworthiness of case studies that evaluate theories [30]. Our case study follows a naturalistic research paradigm as we evaluate the patch-flow measurement method in a real world context. Consequently, the presented trustworthiness criteria are a better fit to evaluate our results than rationalistic criteria (external validity, internal validity, ...) typically used in studies that mine software repositories.

6.3.1 Credibility. Credibility is the degree to which we can establish confidence in the truth of our findings in the context of the inquiry [11]. For ensuring credibility, we applied two techniques:

We performed intense peer debriefing [11]: We intensively discussed this work and gathered feedback from two colleagues within our research group and in an informal setting with external researchers and practitioners. We performed a writer's workshop [10] regarding an earlier draft of this paper with three researchers.

We performed extensive member checks [11]: As described in section 4.2.1, we iteratively performed unstructured interviews and a workshop to support our interpretation of the patch-flow data. During interview I3 and workshop $W3$, we reflected our findings and interpretations to employees of the studied organizations to discuss our findings and receive feedback which we incorporated into this paper.

6.3.2 Dependability. Dependability is the degree of stability of the findings and traceability from collected data to the findings. We established dependability by providing an audit trail [11] containing a log of all (unmodified) data with information on how it was received or measured, when, and by whom. It contains notes for all interviews and workshops and a journal of the analysis process.

6.3.3 Confirmability. Confirmability is the degree to which we are neutral towards the inquiry and might bias the findings [11]. As we are the authors presenting the patch-flow measurement method, we are at risk to overstate the relevance, viability, and usefulness of our method. We addressed this risk using the tactics described in section 6.3.1 regarding credibility.

6.3.4 Transferability. Transferability is the degree to which findings of our inquiry hold validity in other contexts. For our evaluating case study, we selected an established organization that is running IS projects. The setup of our study does not allow us to draw immediate conclusions on whether our patch-flow measurement method is applicable and useful in other contexts. However, we did not find indication that the patch-flow measurement method cannot be applied in other established organizations using IS.

6.4 Future Research

We suggest further research applying the patch-flow measurement method:

Additional case studies. So far, IS case studies collected primary qualitative data. We recommend to perform case study research that not only collects qualitative data on IS but also analyzes the resulting patch-flow. In this paper, we intentionally used our case study only to evaluate the patch-flow measurement method. We suggest that future case studies analyze the patch-flow for theory building, for example to analyze which IS practices trigger or support what volume of contributions from what parties in an organization.

Inner source benchmarks. It is unclear how much IS collaboration happens in organizations. We recommend future research to provide benchmarks of IS collaboration by applying the patch-flow measurement method to a representative sample of software companies. Researchers could use such a benchmark to locate the organizations they study on a map and discuss the generalizability of their findings on a solid basis. Practitioners could gain insights on the maturity and success of their IS initiatives.

Derived metrics. The patch-flow measurement method provides primitive data about the IS collaboration. We encourage researchers to use this data to theorize and validate metrics for practitioners' information needs regarding IS model (e.g. a management accounting model for IS contributions, models for evaluating the performance of IS projects, incentive systems for IS, etc.).

7 CONCLUSION

In this paper, we present the patch-flow measurement method – the first method to measure IS collaboration. It measures flow of code contributions within an organization.

We evaluate our patch-flow measurement method using case study research. In the case study organization, we found significant high patch-flow. This indicates high relevance of the patch-flow phenomenon and justifies the method research presented in this paper. Using case study research, we evaluate the viability of the patch-flow measurement method. We found our method viable to measure the required patch-flow data. We demonstrate the usefulness of the patch-flow data and graphs and found that they are capable to express IS collaboration in the studied multi-industry organization.

We recommend further research based on our method including additional case studies and benchmarks of IS collaboration. We believe our method is of interest to researchers and practitioners seeking to understand IS collaboration within an organization.

8 ACKNOWLEDGMENTS

This work was partially funded by DFG grant 382466185 and two industry grants to the Open Source Research Group at Friedrich-Alexander-Universität. We thank our contacts at the studied organization for their help in collecting and interpreting the patch-flow data, Minghui Zhou for the helpful discussion and feedback, as well as Andreas Bauer, Nikolay Harutyunyan, Andreas Kaufmann, and Daniel Knogl for improving this paper in a writer's workshop, and Sebastian Duda and Julia Werner for improving it with their feedback.

REFERENCES

[1] Andrew Begel, Yit Phang Khoo, and Thomas Zimmermann. 2010. Codebook: discovering and exploiting relationships in software repositories. In *Software Engineering, 2010 ACM/IEEE 32nd International Conference on*, Vol. 1. IEEE, 125–134.

[2] Maximilian Capraro and Dirk Riehle. 2017. Inner source definition, benefits, and challenges. *ACM Computing Surveys (CSUR)* 49, 4 (2017), 67.

[3] Gabriella CB Costa, Francisco Santana, Andréa M Magdaleno, and Cláudia ML Werner. 2014. Monitoring Collaboration in Software Processes Using Social Networks. In *CYTED-RITOS International Workshop on Groupware*. Springer, 89–96.

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

[5] Kevin Crowston, Kangning Wei, James Howison, and Andrea Wiggins. 2008. Free/Libre Open-source Software Development: What We Know and What We Do Not Know. *ACM Comput. Surv.* 44, 2, Article 7 (March 2008), 35 pages. https://doi.org/10.1145/2089125.2089127

[6] Daniela S Cruzes and Tore Dyba. 2011. Recommended steps for thematic synthesis in software engineering. In *Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium on*. IEEE, 275–284.

[7] Jamie Dinkelacker, Pankaj K Garg, Rob Miller, and Dean Nelson. 2002. Progressive open source. In *Proceedings of the 24th International Conference on Software Engineering*. ACM, 177–184. http://doi.acm.org/10.1145/581339.581363

[8] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1994. Design patterns: elements of reusable object-oriented software. (1994).

[9] Georgios Gousios, Eirini Kalliamvakou, and Diomidis Spinellis. 2008. Measuring developer contribution from software repository data. In *Proceedings of the 2008 international working conference on Mining software repositories*. ACM, 129–132.

[10] Hillside Group. 2010. How to Hold a Writers Workshop. (2010). Last retrieved in January 2018, http://hillside.net/conferences/plop/235-how-to-hold-a-writers-workshop. Publication year estimated using http://web.archive.org.

[11] Egon G Guba. 1981. Criteria for assessing the trustworthiness of naturalistic inquiries. *Educational Technology Research and Development* 29, 2 (1981), 75–91.

[12] Vijay K. Gurbani, Anita Garvert, and James D. Herbsleb. 2006. A Case Study of a Corporate Open Source Development Model. In *Proceedings of the 28th International Conference on Software Engineering (ICSE '06)*. ACM, New York, NY, USA, 472–481. https://doi.org/10.1145/1134285.1134352

[13] Vijay K. Gurbani, Anita Garvert, and James D. Herbsleb. 2010. Managing a Corporate Open Source Software Asset. *Commun. ACM* 53, 2 (Feb. 2010), 155–159. https://doi.org/10.1145/1646353.1646392

[14] Anja Guzzi, Andrew Begel, Jessica K Miller, and Krishna Nareddy. 2012. Facilitating enterprise software developer communication with CARES. In *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*. IEEE, 527–536.

[15] Mitchell Joblin, Wolfgang Mauerer, Sven Apel, Janet Siegmund, and Dirk Riehle. 2015. From developer networks to verified communities: a fine-grained approach. In *Proceedings of the 37th International Conference on Software Engineering-Volume 1*. IEEE Press, 563–573.

[16] Eirini Kalliamvakou, Georgios Gousios, Diomidis Spinellis, and Nancy Pouloudi. 2009. Measuring Developer Contribution From Software Repository Data. *MCIS* 2009 (2009), 4th.

[17] Juho Lindman, Matti Rossi, and Pentti Marttiin. 2008. Applying Open Source Development Practices Inside a Company. In *Open Source Development, Communities and Quality*, Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, and Giancarlo Succi (Eds.). IFIP – The International Federation for Information Processing, Vol. 275. Springer US, 381–387. https://doi.org/10.1007/978-0-387-09684-1_36

[18] Gregory Madey, Vincent Freeh, and Renee Tynan. 2002. The open source software development phenomenon: An analysis based on social network theory. *AMCIS 2002 Proceedings* (2002), 247.

[19] Ken Martin and Bill Hoffman. 2007. An Open Source Approach to Developing Software in a Small Organization. *Software, IEEE* 24, 1 (Jan 2007), 46–53. https://doi.org/10.1109/MS.2007.5

[20] Catharina Melian and Magnus Mähring. 2008. Lost and Gained in Translation: Adoption of Open Source Software Development at Hewlett-Packard. In *Open Source Development, Communities and Quality*, Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, and Giancarlo Succi (Eds.). IFIP – The International Federation for Information Processing, Vol. 275. Springer US, 93–104. https://doi.org/10.1007/978-0-387-09684-1_8

[21] Andrew Meneely and Laurie Williams. 2011. Socio-technical developer networks: Should we trust our measurements?. In *Proceedings of the 33rd International Conference on Software Engineering*. ACM, 281–290.

[22] Andreas Neus and Philipp Scherf. 2005. Opening minds: Cultural change with the introduction of open-source collaboration methods. *IBM Systems Journal* 44, 2 (2005), 215–225. https://doi.org/10.1147/sj.442.0215

[23] Andy Oram. 2015. *Getting started with inner source*. O'Reilly Media, Inc.

[24] Tim O'Reilly. 2000. Archived email discussion on Open Source and OpenGL. (2000). Last retrieved in January 2018, http://archive.oreilly.com/pub/a/oreilly/ask_tim/2000/opengl_1200.html.

[25] Martin Pinzger, Nachiappan Nagappan, and Brendan Murphy. 2008. Can developer-module networks predict failures?. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*. ACM, 2–12.

[26] Dirk Riehle. 2015. The five stages of open source volunteering. In *Crowdsourcing*. Springer, 25–38.

[27] Dirk Riehle, John Ellenberger, Tamir Menahem, Boris Mikhailovski, Yuri Natchetoi, Barak Naveh, and Thomas Odenwald. 2009. Open collaboration within corporations using software forges. *Software, IEEE* 26, 2 (2009), 52–58.

[28] Michael Schwind and Christian Wegmann. 2008. SVNNAT: Measuring collaboration in software development networks. In *E-Commerce Technology and the Fifth IEEE Conference on Enterprise Computing, E-Commerce and E-Services, 2008 10th IEEE Conference on*. IEEE, 97–104.

[29] Phillip Smith and Chris Garber-Brown. 2007. Traveling the Open Road: Using Open Source Practices to Transform Our Organization. In *Agile Conference (AGILE), 2007*. 156–161. https://doi.org/10.1109/AGILE.2007.65

[30] Klaas-Jan Stol, Paris Avgeriou, Muhammad Ali Babar, Yan Lucas, and Brian Fitzgerald. 2014. Key Factors for Adopting Inner Source. *ACM Trans. Softw. Eng. Methodol.* 23, 2, Article 18 (April 2014), 35 pages. https://doi.org/10.1145/2533685

[31] Klaas-Jan Stol, Muhammad Ali Babar, Paris Avgeriou, and Brian Fitzgerald. 2011. A comparative study of challenges in integrating Open Source Software and Inner Source Software. *Information and Software Technology* 53, 12 (2011), 1319–1336.

[32] Richard Torkar, Pau Minoves, and Janina Garrigós. 2011. Adopting free/libre/open source software practices, techniques and methods for industrial use. *Journal of the Association for Information Systems* 12, 1 (2011), 88–122.

[33] Yuriy Tymchuk, Andrea Mocci, and Michele Lanza. 2014. Collaboration in open-source projects: Myth or reality?. In *Proceedings of the 11th working conference on mining software repositories*. ACM, 304–307.

[34] Frank van der Linden. 2013. Open Source Practices in Software Product Line Engineering. In *Software Engineering*, Andrea De Lucia and Filomena Ferrucci (Eds.). Lecture Notes in Computer Science, Vol. 7171. Springer Berlin Heidelberg, 216–235. https://doi.org/10.1007/978-3-642-36054-1_8

[35] Frank van der Linden, Björn Lundell, and Pentti Marttiin. 2009. Commodification of Industrial Software: A Case for Open Source. *Software, IEEE* 26, 4 (July 2009), 77–83. https://doi.org/10.1109/MS.2009.88

[36] Padmal Vitharana, Julie King, and Helena Shih Chapman. 2010. Impact of internal open source development on reuse: Participatory reuse in action. *Journal of Management Information Systems* 27, 2 (2010), 277–304.

[37] Robert K Yin. 2013. *Case study research: Design and methods*. Sage publications.