Measuring Inner Source Collaboration

Vermessung von Inner-Source-Zusammenarbeit

Der technischen Fakultät
der Friedrich-Alexander-Universität
Erlangen-Nürnberg

zur

Erlangung des Doktorgrades
Doktor-Ingenieur (Dr.-Ing.)

vorgelegt von

Maximilian Capraro

Als Dissertation genehmigt
von der Technischen Fakultät
der Friedrich-Alexander-Universität Erlangen-Nürnberg

Tag der mündlichen Prüfung:	19. Mai 2020
Vorsitzender des Promotionsorgans:	Prof. Dr.-Ing. Andreas P. Fröba
Gutachter:	Prof. Dr. Dirk Riehle Prof. Dr. Brian Fitzgerald

Abstract

Inner source (IS) is the use of open source software development practices and the establishment of open source-like communities within an organization. The organization may still develop proprietary software but internally opens up its development. IS promises to resolve problems of traditional software development by easing software reuse and enabling parties within an organization to collaborate across organizational boundaries.

However, it is unclear what elements constitute IS (problem I) and how to measure the presence and magnitude of IS collaboration (problem II). The large majority of research articles on IS to date are limited to qualitative results regarding IS. There are yet no quantitative studies on IS collaboration exploring how much IS collaboration takes place or how IS practices affect it (problem III).

We followed a three-phase research approach to address these problems. First, we performed an extensive literature survey and analyzed 43 IS publications. We found that four key elements constitute IS (shared cultural values, open development environment, communities around software, IS-specific scenarios) but that IS programs and projects differ on at least five dimensions (addressing problem I).

Second, we developed the patch-flow method (and a software tool implementing it) for measuring IS collaboration. Patch-flow is the flow of code contributions across organizational boundaries ("silos") such as organizational unit or cost center boundaries. We evaluated the method using case study research with a non-trivial industry organization and found it to be viable and useful to practitioners (addressing problem II).

Third, we performed a multiple-case case study with three large software organizations running a total of five IS program. We identified the used IS practices and the resulting patch-flow. We found patch-flow to exist in all organizations but that only fraction of all code contributions to IS projects constitute patch-flow. We observed that the number of IS practices implemented correlates with the distance of parties involved in collaboration. This indicates that IS is particularly suited to enable collaboration between parties of high distance in an organization (addressing problem III).

This thesis delivers a holistic definition of IS and the first classification framework for IS programs and projects. Researchers can use such a framework to reason about generalizability of their results more precisely. The patch-flow measurement method is the first of its kind to measure and quantify IS collaboration and can serve as a base for further quantitative analyses of IS collaboration. The exploration of the patch-flow in the three industry cases can serve as example and benchmark for practitioners.

Zusammenfassung

Inner Source (IS) ist die Verwendung von Softwareentwicklungspraktiken aus dem Open-Source-Umfeld und die Etablierung von open-source-artigen Communities innerhalb der Grenzen einer Organisation. Die Organisation kann dabei weiterhin proprietäre Software entwickeln und öffnet die Softwareentwicklung lediglich für interne Teilnehmer. IS löst Probleme der traditionellen Softwareentwicklung, in dem es Wiederverwendung vereinfacht und Parteien innerhalb einer Organisation erlaubt, über organisationsinterne Grenzen hinweg, zusammenzuarbeiten.

Allerdings ist unklar, welche Elemente IS definieren (Problem I) und wie man das Vorhandensein und Ausmaß von IS-Zusammenarbeit messen kann (Problem II). Die Mehrheit an existierenden Forschungsartikeln zu IS präsentiert lediglich qualitative Ergebnisse. Es gibt bis jetzt keine quantitativen Studien, die explorieren wie viel IS-Zusammenarbeit vorkommt oder wie IS-Praktiken sie beeinflussen (Problem III).

Wir adressieren diese Probleme mit einem Forschungsansatz in drei Phasen: Im der ersten Phase führen wir ein umfangreiches Literatur-Survey durch und analysieren 43 Publikationen zu IS. Wir stellen vier Schlüsselelemente vor, die IS ausmachen (gemeinsame kulturelle Werte, eine offene Entwicklungsumgebung, Communities um Software, IS-spezfische Szenarios) und zeigen, dass IS-Programme und -Projekte sich in mindestens fünf Dimensionen unterscheiden können (adressiert Problem I).

Im zweiten Schritt entwickelen wir die Patch-Flow Methode (und ein Softwarewerkzeug, dass sie implementiert) um die Vermessung von IS-Zusammenarbeit zu ermöglichen. Patch-Flow ist der Fluss von Codebeiträgen über organisationsinterne Grenzen ("Silos") wie beispielsweise Organisationseinheiten oder Kostenstellen hinweg. Wir evaluieren die Methode mittels Fallstudienforschung in einer nicht-trivialen Organisation und zeigen, dass die Methode in der Praxis umsetzbar und für Praktiker von Nutzen ist (adressiert Problem II).

Im dritten Schritt führen wir eine Fallstudie mit drei großen Softwareorganisationen durch, die insgesamt fünf IS-Programme betreiben. Wir identifizieren verwendete IS-Praktiken und vermessen den Patch-Flow. Patch-flow existiert in allen untersuchten Organisationen, jedoch ist nur ein Bruchteil aller Codebeiträge Patch-Flow. Wir beobachteten, dass die Anzahl der IS-Praktiken mit der Distanz der zusammenarbeitenden Parteien korreliert. Das deutet darauf hin, dass IS besonders geeignet ist, um Zusammenarbeit zwischen Parteien in hoher Distanz zu ermöglichen (adressiert Problem III).

Diese Thesis präsentiert eine umfassende Definition von IS sowie das erste Klassifikationsrahmenwerk für IS-Programme und -Projekte. Forscher können das Rahmenwerk nutzen um die Generalisierbarkeit ihrer Ergebnisse präziser zu diskutieren. Die Patch-Flow Methode ist die erste Methode zur Vermessung und Quantifizierung der IS-Zusammenarbeit. Sie kann als Basis für weitere quantitative Analysen der IS-Zusammenarbeit dienen. Die Exploration des Patch-Flow in drei Organisationen dient als Beispiel und Benchmark für Praktiker.

Contents

Listing of Figures

Listing of Tables

Listing of Acronyms

API	application programming interface
CLI	command line interface
CSV	comma separated value
IS	inner source
LCA	lowest common ancestor
LDAP	lightweight directory access protocol
org. level	organizational level
org. unit	organizational unit
OS	open source
PMC	project management committee
REST	representational state transfer
SCM	source code management
SIP	session initiation protocol
SNA	social network analysis
SPL	software product line
TFS	Team Foundation Server

Acknowledgments

A large number of human beings supported this thesis project with their time, expertise, and encouragement. I am very grateful to every single one of them.

First and foremost, I wish to express my deep gratitude to my thesis advisor Prof. Dirk Riehle who introduced me to inner source, provided continuous guidance, taught me about the software industry and its people, allowed me to pursue this research, and by doing so made this thesis project possible.

I am grateful to Prof. Brian Fitzgerald for being external examiner of this thesis and serving in my promotion committee.

During the last years, I had the pleasure to work in a group comprised of the most extraordinary people: Dr. Ann Barcomb, Andreas Bauer, Andreas Kaufmann, Daniel Knogl, Fariba Bensing, Georg Schwarz, Hannes Dohrn, Julia Krause, Michael Dorner, Dr.-Ing. Nikolay Harutyunyan, and Sebastian Schmid. I am grateful to every single one them for their enormous support and encouragement. Over the years, they each reviewed iterations of the papers that built the foundation for this thesis or even chapters of this thesis itself. They helped to significantly improve my work.

Michael was my office mate and a trusted confidant during the course of my thesis project. Practically daily, we discussed our research, reflected on observations, and hatched ideas. While writing this thesis, his LaTeX help was invaluable. I cannot thank him enough.

I am grateful to Ann and Andreas K. who on countless occasions offered feedback, help, and entertaining sarcasm. I wish to thank Nikolay for repeatedly sharing with me experiences from the final months of his thesis project. This allowed me to navigate the various processes at our university smoothly and focus on the actual research and content work. I thank Andreas B., Andreas K., Hannes, and Georg for the many discussions on software development that helped to improve my measurement software significantly and broadened my horizon.

During the course of this thesis project, researchers outside of our research group or adjunct to it offered help. I am very grateful for the advise and the always constructive feedback, Dr. Klaas-Jan Stol and Prof. Minghui Zhou have offered on multiple occasions during the last

years. I am very thankful to Dr.-Ing. Martin Jung for his feedback that helped to improve the more technical parts of this thesis.

Research on software engineering does not work well without an industry perspective. I wish to thank all of the participants of our case studies as well as the many individuals from other organizations in industry who offered feedback, insights, and inspiration. I am particularly grateful for the countless conversations with Andreas Hadert, Jochen Michel, Martin Stern, Ralph Luber, Siegfried Wach, Dr. Stefan Voget, Thomas Hildebrand, Tobias Winderl, and Zsuzsanna Gnandt that benefited our research and this thesis greatly.

Over the last years an industry community formed about understanding, learning, and teaching inner source principles – the InnerSource Commons. I would like thank all of its members, particularly Cedric Williams, Dr. Daniel Izquierdo Cortázar, Georg Grütter, Johannes Tigges, Dr. Tim Yao and Russel Rutledge, for the countless discussions and the vivid exchange of ideas; and Danese Cooper for launching this community.

I am deeply grateful to Julia Werner for her feedback on data visualization and writing, her support and inspiration during the course of this thesis project. Last, but absolutely not least, I wish to thank my parents Annegret and Franz, my family, and my friends for their endless support.

I acknowledge Jordan Suchow for providing the template that was used to typeset this thesis (*https://github.com/suchow/dissertate*) and Nikolay Harutyunyan and Andreas Kaufmann for modifying this template according to the requirements of our university. This work was partially funded by DFG grant 382466185, three industry grants by undisclosed organizations, and an industry grant by Black Duck Software Inc. to the Professorship of Open Source Software at Friedrich-Alexander-Universität Erlangen-Nürnberg.

Related Publications

Throughout this thesis, the third person plural (“*we*”) is used. That is common courtesy given the support that was acknowledged in the previous section. However, in this section, the first person singular (“*I*”) is used to allow for a clear distinction between the contributions of the author of this thesis and others.

My colleagues and I published or intend to publish partial results of this thesis. The following articles are related to this thesis:

Capraro and Riehle 2017 *Inner source definition, benefits, and challenges.* For chapter 2 of this thesis, I reused the article’s components regarding the IS taxonomy. For this thesis, I extended them with a more detailed discussion of the approach and performed changes to the text enhancing its readability. Appendix A is copied verbatim from the article with minor changes.

Capraro et al. 2018 *Patch-flow method for measuring inner source collaboration.* Chapter 3 is copied verbatim from this article. For this thesis, I extended the text with a clearer distinction between contribution-flow and patch-flow phenomena, added a discussion on how patch-flow can be measured for different classes of IS programs and projects, and performed minor changes.

not yet published *How inner source practices affect inner source collaboration.* Chapter 5 is based on the findings of this study and the current draft of a manuscript reporting about them.

For all three articles, I was the leading author and the main contributor to the research design, execution, and writing.

For Capraro and Riehle (2017), Dirk Riehle contributed by supporting the literature selection and providing significant feedback on the research design and revisions of the article. For

Capraro et al. (2018), Michael Dorner and Dirk Riehle contributed by providing significant feedback on the revisions of the manuscript. Michael Dorner contributed to the data collection and interpretation. For the current version of the not yet published manuscript, Michael Dorner and Dirk Riehle contributed by providing significant feedback on the research design and revisions of the manuscript. Michael Dorner contributed to the data collection in two of the three cases.

1

Introduction

Open source (OS) software - that is software provided to the world under an open source license, free for everyone to use and modify - has become a key pillar of today's software industry. OS software tools are used as part of software development processes (Fitzgerald 2006), OS components are integrated into proprietary software products, and often OS infrastructure components deliver and run these products (Franch Gutiérrez et al. 2013). Organizations invest significant effort in governing their use and contributions to OS (Harutyunyan and Riehle 2019; Harutyunyan 2019) or use it as part of their business strategy (Riehle 2007, 2009, 2012). OS is recognized to be capable of delivering high quality software (Crowston et al. 2012).

The software industry has shown a significant interest in using not only from OS' outcomes (the software components and tools) but also adopting software development (software development) practices that are typically exercised in the OS context (Stol et al. 2014). The use of OS software development practices and the establishment of OS-like culture within organizations is called *inner source* (IS). Practitioners expect a variety of benefits from IS adoption (see appendix A). 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 (Capraro and Riehle 2017).

While, IS software development is similar to and shares attributes with OS software development, IS opens up the development only internally within the environment of one organization (Dinkelacker et al. 2002). Organizations can continue to develop proprietary software and do not need to provide it to the world under an OS license. However, they establish OS-like communities consisting only of individuals employed by the organization. IS is believed to significantly increase the software development efficiency and lead to higher quality software components. It leads to "participatory reuse" (Vitharana et al. 2010) where (firm-internal) users of a software component become contributors to it.

The next sections will motivate and introduce the research questions answered in this thesis (section 1.1) and give basic IS definitions and a brief introduction of IS's mechanics (section 1.2).

1.1 Motivation, Research Questions, Contributions

Despite the interest of both research and practitioner communities and first publications dating back nearly 20 years at time of this writing (O'Reilly 2000; Dinkelacker and Garg 2001; Dinkelacker et al. 2002), we consider IS to still be an emerging phenomenon. Certain base research has not yet been performed, leaving researchers with a weak foundation for their work (and practitioners with a weak foundation for IS adoption):

1. There is no established taxonomy defining IS thoroughly.
2. There is no established operational definition or method to measure IS.
3. Prior work consists of primarily qualitative studies. It is unclear how much IS collaboration takes place and how different IS practices affect it.

This thesis reports on three studies resolving these problems. First, we present a literature survey resulting in a model of key elements of IS and a classification framework for IS programs and projects giving a thorough definition of IS. Second, we describe the patch-flow measurement method for measuring IS collaboration, report on a single-case case study evaluating its viability and utility, and present a software tool for measuring patch-flow. Third, we present multiple-case case study research applying the patch-flow measurement method in three organizations to deliver the first quantitative study on IS describing the magnitude of IS collaboration. As part of this study, we build a theory on how IS practices affect IS collaboration.

The following subsections detail the motivation, research questions, and contributions of each of these studys.

1.1.1 Inner Source Taxonomy

A steady stream of of scientific literature and practitioner reports indicates the interest in IS. However, the area lacks a systematic arrangement of prior research work: No consistent taxonomy has been presented yet. It is not clear which general elements constitute IS. Differences between IS programs were studied only in the context of a few selected organizations. The absence of models and theories that have validity for more than a few organizations leaves researchers with a weak foundation for further research. It creates uncertainty and the risk that the term IS is used with ambiguous meanings or varying understandings.

To this end, this thesis answers the following research questions:

- RQ1: What are the elements of IS software development?
- RQ2: How do different IS implementations differ from one another?

We answer the research questions by assessing the state of the research: We performed a literature survey considering a total 43 scientific publications plus additional material. We analyzed the materials using the inductive theory generation method (Thomas 2006).

To this end, this thesis contributes the following:

- A taxonomy of IS, including ...
  - a theoretical model of elements constituting IS
  - a classification framework for IS programs and projects as well as its application to known instances

We found that four key elements constitute IS (an open development environment, shared cultural values, communities around software, and IS collaboration scenarios). However, IS programs can differ on at least three dimensions (prevalence of IS, market mechanisms, degree of self-organization) and IS projects on at least two dimensions (governance, project objective).

Our model and framework provides a more solid foundation for further research. Researchers can use our model of IS elements as a lens to discuss the IS programs they study and the classification framework to reason about the generalizability of IS their findings.

1.1.2 Method for Measuring Inner Source Collaboration

There is yet no method to measure IS collaboration within an organization.Such a method can benefit 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.

To this end, this thesis answers the following research question:

- RQ3: How to measure IS collaboration within a software developing organization?

We answer the research question by formalizing the patch-flow method and evaluating its relevance, viability, and usefulness using case study research (Yin 2013). Patch-flow is the flow of code contributions across organizational boundaries such as project or cost center boundaries. The word patch is historically derived from the “patch files” 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). When measuring patch-flow, it is not sufficient to simply count patches over time. One must address the organizational structure contextual to the patch.

To this end, this thesis contributes the following:

- A definition of the patch-flow phenomenon
- The 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
- An introduction and discussion of the patch-flow crawler – a tool for measuring patch-flow

Our case study demonstrated the viability of the patch-flow measurement model. We found the method viable to measure the patch-flow data and that the patch-flow data and visualizations are capable and useful to express IS collaboration in the studied organization. We observed significant patch-flow, indicating high relevance of the patch-flow phenomenon and thus our method research.

We believe our method to be of interest to both researchers and practitioners seeking to understand IS collaboration within an organization.

1.1.3 Influence of Inner Source Practices on Collaboration

In absence of a method to measure IS collaboration, the majority of scientific literature presented qualitative results such as case study reports (for example Dinkelacker et al. 2002; Gurbani et al. 2006; Stol et al. 2014; Riehle et al. 2016) and taxonomies or qualitative models regarding IS (for example Stol et al. 2011, 2014; Gaughan et al. 2007). There is no study yet quantifying the magnitude or structure of IS collaboration or exploring the relationship between exercised IS practices and resulting IS collaboration.

Learning about the magnitude of IS collaboration is relevant for IS practitioners and researchers because it tells how much IS collaboration is to be expected in organizations. The magnitude indicates the relevance of research on IS proxied by IS's possible impact on industry organizations. Understanding how implemented IS practices affect IS collaboration is of interest to to primarily practitioners. It indicates whether the cost of adopting specific IS practices is justified because it leads to a different magnitude or properties of IS collaboration.

To this end, this thesis answers the following research questions:

- RQ4: What is the magnitude of IS collaboration in organizations?
- RQ5: How do IS practices affect IS collaboration?

To answer these research questions, we performed a multiple-case case study in three software developing organizations running five IS programs. We collected and analyzed qualitative data to identify the IS practices exercised and the patch-flow (as a proxy for IS collaboration) in the context of each IS program. We identified correlations between the observed IS practices and patch-flow and theorized about relationships between them.

To this end, this thesis delivers the following:

- A multiple-case case study with three software developing organizations, including ...
    - a discussion of how IS is implemented in five IS programs in the three case organizations
    - an in-depth analysis of the case organizations' patch-flow

- A theory (consisting of four hypotheses) on how IS practices affect IS collaboration

Patch-flow and thus IS collaboration occurred in all IS programs. Its magnitude and structure differed significantly. We theorize that a higher number of IS practices alone does not necessarily result in more IS collaboration, but that significant IS collaboration can occur with crude and rudimentary IS practices if there is a strong need to collaborate on a specific component. We found a correlation between IS practices and the distance of organizational units (org. units) collaborating. We theorize that IS practices enable collaboration among org. units in high organizational distance (between whom no collaboration would take place otherwise).

1.2 Inner Source Definitions

The term IS was coined by O'Reilly (2000). We define IS as follows:

> *Inner source* (IS) is the use of OS software development practices and the establishment of an OS-like culture within organizations.

In IS, selected software components are made available as IS projects. We define the term as follows:

> An *IS project* is a software project with the goal to develop and maintain IS software.

IS projects are like OS projects in that they do not have a defined end date. Like in OS, the name of the project is often also used to address the IS component. The software developed by IS projects is IS software. Stol et al. (2011) define the term as follows:

> *IS software* is the "software product [or component] that is developed within an IS context".

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 org. 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 the code base (Gurbani et al. 2006; Riehle et al. 2009). We define the term as follows:

> A *committer* is an individual with write (“commit”) privileges to a project’s code base.

An IS project typically has one or many committers.

Participants in IS projects communicate openly: Every individual within the company can read and participate in discussions (Neus and Scherf 2005). Open communication should not only be public within the organization but also archived, written, and complete (Riehle 2015). A common tool for exercising open communication is a mailing list (van der Linden 2013) or the features for discussing issues in software forges like GitHub and Gitlab.

To support IS adoption, organizations can establish an IS program. We define the term as follows:

> An *IS program* is a coordinated effort of an organization to run and maintain one or multiple IS projects.

Dinkelacker et al. (2002) first used the term while discussing HP’s corporate source and collaborative development programs. The collection of all IS software components that are developed within the projects of an IS program for its IS portfolio. We define the term as follows:

> An *IS portfolio* is the set of all IS software components that are developed and maintained as part of an IS program.

The project-specific and program-wide perspective differ significantly from each other. The view on an IS program as a whole focuses on an organization’s complete IS landscape. A project-specific view is focused on the surroundings of one specific project, the involved parties, and their interest. Distinguishing between these perspectives but also considering both is crucial for understanding an organization’s IS efforts.

1.3 Thesis Structure

The remainder of this thesis is structured as follows.

- **Chapter 2 (IS taxonomy)** reports on a literature survey study resulting in a taxonomy of IS. It extends the definitions given as part of the introduction by presenting a model of key elements that constitute IS and a classification framework for IS programs and projects.

- **Chapter 3 (patch-flow measurement method)** introduces the patch-flow measurement method for measuring IS collaboration and reports on our single-case case study research to evaluate the method's relevance, viability, and usefulness to practitioners and researchers.

- **Chapter 4 (patch-flow crawler)** introduces and discusses the patch-flow crawler - a software tool we developed automate the process of patch-flow measurement. We discuss requirements towards the patch-flow crawler as well as its design and implementation.

- **Chapter 5 (multiple-case case study on IS collaboration)** reports on a multiple-case case study with three large software organizations (running five IS programs) where we investigated the relationship between exercised IS practices and resulting IS collaboration.

- **Chapter 6 (discussion and conclusion)** closes this thesis by laying out future research, highlighting insights relevant to the practitioner community, and presenting our conclusions.

2 Inner Source Taxonomy

A steady stream of scientific literature (and gray literature such as blog posts and magazine articles) since 2001 indicates a vivid interest in inner source (IS) by both researcher and practitioner communities. The majority of scientific publications so far present case studies of IS in the context of one or a few organizations. However, the research area lacks a systematic arrangement of prior research work: No consistent taxonomy has been presented yet. It is not clear, which general elements constitute IS or what differences exist between IS programs.

The absence of generalizable models and definitions that have validity for more than a few organizations leaves researchers with a weak foundation for further research. It creates uncertainty and the risk that the term IS is used with ambiguous meanings or varying understandings. To this end, this chapter answers the following research question:

- RQ1: What are the elements of IS software development?
- RQ2: How do different IS implementations differ from one another?

We answer the research questions by performing a literature survey for which we considered 43 scientific publications plus additional gray literature. This chapter contributes a taxonomy of IS consisting of the following:

- A theoretical model of elements constituting IS

- A classification framework for IS programs and projects as well as its application to known instances

The remainder of this chapter is structured as follows: Section 2.1 discusses prior work related to our study. Section 2.2 presents our approach for literature selection and analysis. The subsequent sections present the results from our literature analysis: a model of elements that constitute IS (section 2.3) and a classification framework of IS programs and projects (section 2.4) as well as its application to known instances of IS (section 2.5). Section 2.6 closes the chapter with our conclusions.

2.1 Related Work

Naturally, the results of a literature survey study on IS contain, summarize, and rearrange publications related to IS and our research objectives. However, some publications presented similar contributions to ours. In the next paragraphs, we briefly discuss the relationship of this prior work to the results we present.

2.1.1 Elements of Inner Source

Sharma et al. (2002) analyzed open source (OS) communities and derived a framework for creating IS communities. Their framework suggests that community building, governance of the community, and community infrastructure are critical to IS adoption. Stol et al. (2014) presented a model of nine key factors regarding *software products*, *practices & tools*, and *organization & community* that need to be considered for successful IS adoption. Contrary to the models by Sharma et al. (2002) and Stol et al. (2014), our model does not aim to explain IS adoption but to define IS and its elements. Stol et al. (2014) also discussed attributes that characterize IS. Our model of elements of IS integrates some of their attributes (for example, "universal access to development artifacts" and "informal communication" as part of the open environment element). However, our model presents a more extensive and detailed account of IS elements.

Other models also aim to characterize IS itself and not only the IS adoption. Vitharana et al. (2010) present a theoretical model of IS based on a case study within IBM's IS program called Community Source. Their paper shows that IS adoption leads to an open development infrastructure, information sharing, and broader community skills which finally results in enhanced reuse. The model does not discuss which elements constitute IS. Gaughan et al. (2009) introduced a model describing IS based on prerequisites, challenges, benefits, practices of IS regarding developed software products and the processes. The model does not present the elements IS is composed of.

Each of the mentioned models delivers a specific perspective on IS but none describes which elements constitute IS or distinguish it from other development approaches. In section 2.3, we will present a theoretical model closing this gap.

2.1.2 Classification Framework

We are not the first to present a classification framework of IS. Gurbani et al. (2010) introduced a classification model differentiating between *infrastructure-based* IS where a central group provides IS infrastructure and parties within the organization can run their own IS projects and *project-specific IS* where one often strategically important IS project is developed. Stol et al. (2014) classified IS programs of nine organizations according to this model. They found infrastructure-based IS to be more prevalent than project-specific IS.

Lindman et al. (2013) introduced a model distinguishing between *private-market IS* where organizational units (org. units) can place software components for sale at an internal IS market and *local-library IS* where the use of components is free.

We integrated the models of Gurbani et al. (2010) and Lindman et al. (2013) into our IS classification framework in section 2.4.

2.2 Research Approach

We followed a two phase research approach. First, in the literature selection phase, we identified relevant IS literature. Second, in the literature analysis phase, we analyzed and systematically arranged the literature. We did not execute these phases strictly sequentially but repeated literature selection multiple times after starting the literature analysis to also cover newer publications.

2.2.1 Literature Selection

For literature selection, we followed a five step approach.

Step 1) Identify papers using keyword search For identifying literature, we performed a keyword search using Google Scholar, the ACM digital library, and IEEE Xplore. We utilized a variety of keywords, phrases, and combinations:

- Inner source, internal open source, firm-internal open source, in-house open source, inside open source, hybrid open source (regarding coordinated IS efforts)

- Social collaboration, open collaboration, social coding (broader context of open collaboration within organizations)

We focused on publications in the field of computer science, information systems, and software engineering. We quicky found that the body of literature regarding IS was significantly smaller compared to more established software development methods. As a consequence, it was not necessary to define narrower keywords.

In their guidelines for literature reviews, Webster and Watson (2002) argue that "major contributions are likely to be in the leading journals" and thus suggest "to start with them" when selecting literature. We explicitly did *not* follow this suggestion. For an emerging and not yet mature topic as IS, we assumed that a majority of literature might not be published in top software engineering journals and conferences but in lower ranked peer-reviewed venues.

Step 2) Decide on inclusion, exclusion Subsequently, we manually determined whether to include or exclude publications identified in step (1) by investigating whether they address IS or not. We did so by reading the title, the abstract, or the full publication.

Step 3) Perform "snowballing" For each included paper, we performed "snowballing" (Wohlin 2014): We transitively checked all forward and backward references of each included paper and decided whether to include them as well (see step 2) until the population of literature was exhausted.

Step 4) Check authors' publications For all included publications, we searched for other publications by the same authors and decided whether they could be included (see step 2) and repeated snowballing where necessary (see step 3).

Figure 2.1: Inner source related publications over time

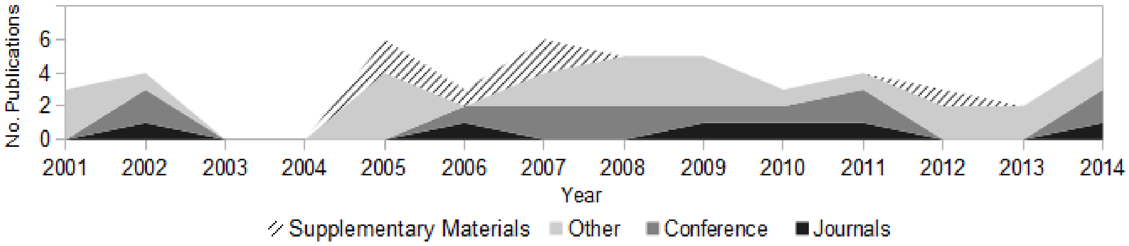
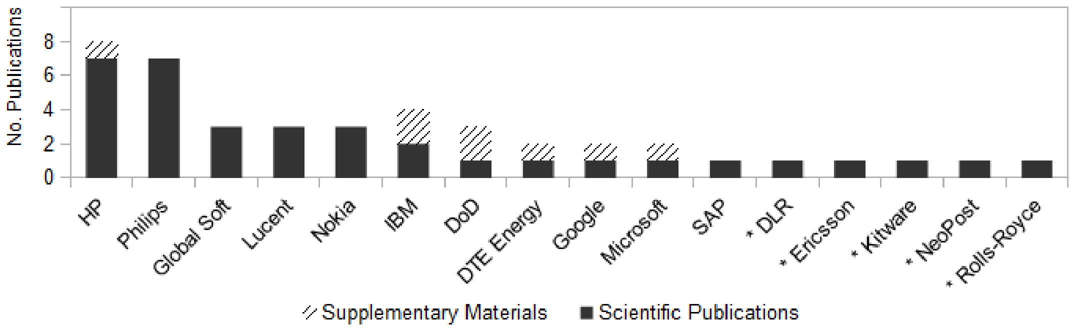


Figure 2.2: Inner source related publications by organization



Step 5) Ask for expert review   Webster and Watson (2002) discuss there is a high chance to miss articles in literature review but that such missing articles are "likely to be identified by colleagues who read your paper". We asked two individuals outside our research group who performed significant research on IS to review a draft of the paper that is the foundation for this chapter and incorporated their feedback.

2.2.2 Resulting Literature

Our literature collection resulted in a total of 43 publications regarding IS. We identified conference papers (12), book chapters (7), journal articles (6), technical reports (5), workshop papers (3), and articles published in other venues (10). Eight non-scientific articles (blog entries, magazines) were cited by the publications or found relevant for this survey. Figure 2.1 shows the number of IS publications in journals, conferences, and other venues per year. For completeness, it also shows non-scientific literature as supplementary materials. A steady stream of publications can be observed, with the exception of 2003 and 2004, when nothing was contributed. A majority of the identified literature were case studies or reports regarding specific IS programs (27). The surveyed literature reports about at least 16 organizations utilizing IS. Figure 2.2 shows the amount of literature regarding each organization ordered by the amount of scientific publications. Supplementary material and non-scientific publications are indicated by hachured filling. Only six organizations were discussed more than once in scientific literature. GlobalSoft is a pseudonym for one organization whose name was not disclosed by the authors of the surveyed papers.

Five organizations did not have a significant impact on our survey study and are marked with an asterisk (*). Literature only allowed superficial insights into the IS programs at DLR (Schreiber et al. 2014) and Kitware (Martin and Hoffman 2007). At Neopost and Rolls-Royce, no in-depth case studies were performed. Much more, Stol et al. (2014) performed evaluating case studies at these organizations evaluating nine factors of IS adoption they have identified. Regarding Ericsson, Torkar et al. (2011) only analyzed the process alignment of Ericsson development and OS processes in preparation of IS adoption. For further analyses we focus on the organizations (and their respective IS programs) that are not marked with an asterisk. Table 2.1 summarizes the resulting 13 IS programs that we considered for the survey.

Table 2.1: Overview of organizations using inner source

Org.	**Program**	**Summary**
DTE Energy	(No name)	DTE Energy adopted IS. They opened up all their source code and made it accessible to all developers within the organization. Through trainings and gatherings a community that transcended organizational units was formed (Smith and Garber-Brown 2007).
Google	(No name)	All source code at Google shares a single repository. All developers have read access and are encouraged to contribute (Whittaker et al. 2012). Developers can utilize 20% of their time for projects out of their immediate scope which can lead to volunteering like in open source (Google-Blog 2006).
HP	Collaborative Development Program (CDP)	The CDP is an IS program offering a set of web-based collaboration tools. CDP is not only designed to enable collaboration of HP employees but also collaboration with partners and contractors of HP (Dinkelacker et al. 2002). Dinkelacker et al. (2002) coined the term controlled source to describe a development model where business partners collaborate on proprietary software using open source practices. We refer to this model as partner source. In this thesis, we will only discuss inner source but not partner source.

Table 2.1: Overview of organizations using inner source - continued

Org.	Program	Summary
HP	Corporate Source (CS)	CS is an IS program which was initially founded to extend HP's research community into the organizational units at HP that develop products. A searchable intranet website makes IS projects within the IS portfolio searchable for developers (Dinkelacker et al. 2002).
IBM	Community Source (CMS)	CMS is a IS program program based around a set of provided tools (project hosting, software repositories, mailing lists, bug trackers). It makes selected confidential software components available to developers that are not directly involved with the projects. Manager approval is needed for taking part (Vitharana et al. 2010; Fox 2007).
IBM	IBM Internal Open Source Bazaar (IIOSB)	IIOSB is built on the same set of tools as IBM's community source but makes source code available to all developers within the organization without the need of a manager's approval. Developers are empowered to contribute to projects outside their scope with little effort (Vitharana et al. 2010).
Lucent	(No name)	A developer at Lucent implemented a Session Initiation Protocol (SIP) server. SIP is a protocol for communications and telephony. The developer stepwise made the implementation and later the source code available to developers within the organization (Gurbani et al. 2006). An internal group was founded to steer the company-wide collaboration regarding the project (Gurbani et al. 2010).
Microsoft	CodeBox	CodeBox is a software forge and IS program at Microsoft (Asay 2007). Some groups of Microsoft's research and development departments use CodeBox as primary development infrastructure and project communities have formed around the developed software (Microsoft 2008). A similar software forge is publicly available under the name CodePlex.
Nokia	iSource	Nokia's iSource is a firm internal software forge and IS program. The underlying forge is based on a fork of the sourceforge.net software. All Nokia engineers can participate in iSource which is counting over 100 IS projects as of 2008 (Lindman et al. 2008).

Table 2.1: Overview of organizations using inner source - continued

Org.	Program	Summary
Philips	(No name)	Philips applies IS as part of their product line engineering for medical devices. General implementations like an implementation of the DICOM medical imaging standard are developed using an IS approach (van der Linden 2009). Before adopting IS, Philips implemented market mechanisms that forced reusers to pay a fee for using a component (Wesselius 2008).
SAP	Firm Internal Software Forge	SAP's firm internal software forge was created by internal research departments with the main goal of enhancing the research to product transfer and embracing the principles of open collaboration within SAP. As of June 2007 over 400 projects were reported to use the SAP internal software forge (Riehle et al. 2009).
Department of Defense	Forge.mil	Forge.mil is a software forge and IS program used by the United States Department of Defense to collaborate internally and with their partners. Forge.mil allows collaboration with employees within or with partners outside the organization (Martin and Lippold 2011). It is therefore an instance of inner source and partner source.
GlobalSoft	(No name)	GlobalSoft adopted IS to in the context of their software product line engineering (Höst et al. 2014). The name is a pseudonym of a case organization that was not disclosed.

2.2.3 Literature Analysis

We analyzed the literature to build our theoretical model of elements that constitute IS and to develop the classification framework for IS.

Elements of inner source

One of this chapter's contribution is a theoretical model of elements that constitute IS. To identify these elements, we utilized inductive theory generation (a method for analysis of qualitative data) by Thomas (2006) using the surveyed literature as input data. Inductive theory generation required us to code segments within all considered publications into categories. This process is called a 'coding process'.

The coding process suggested by Thomas (2006) consists of five steps: Initially, we familiarized ourselves with the literature (1) and subsequently identified and labeled text segments with categories related to our objective of identifying elements that constitute IS (2). These categories then were grouped by common themes (3). Finally, we reduced the amount of categories by reducing overlap and redundancy among them (4) and discarded categories with little importance (5). We used the software tool MaxQDA* for this coding.

This process resulted in a hierarchical arrangement of codes (a so called 'code system') with four top-level categories, lower-level categories belonging to these categories, and links expressing which text segments were labeled to belong to these categories. We transfered this code system into our model of elements of IS (section 2.3). The resulting model is a qualitative model. Contrary to quantitative models that represent or predict phenomena mathematically, qualitative models express concepts and their relationships.

Classification framework

This chapter contributes a classification framework for IS programs and projects. During the third and fourth phases of the coding process regarding the elements of IS, we found categories that contradicted each other. For example, some organizations internally opened all their source code for IS, while others selected specific components to be inner-sourced. These categories were obviously not fit to describe general elements of IS. However, they can serve to describe variation points of IS. We integrated both the contradicting categories and classifications from known literature into our classification framework presented in section 2.4.

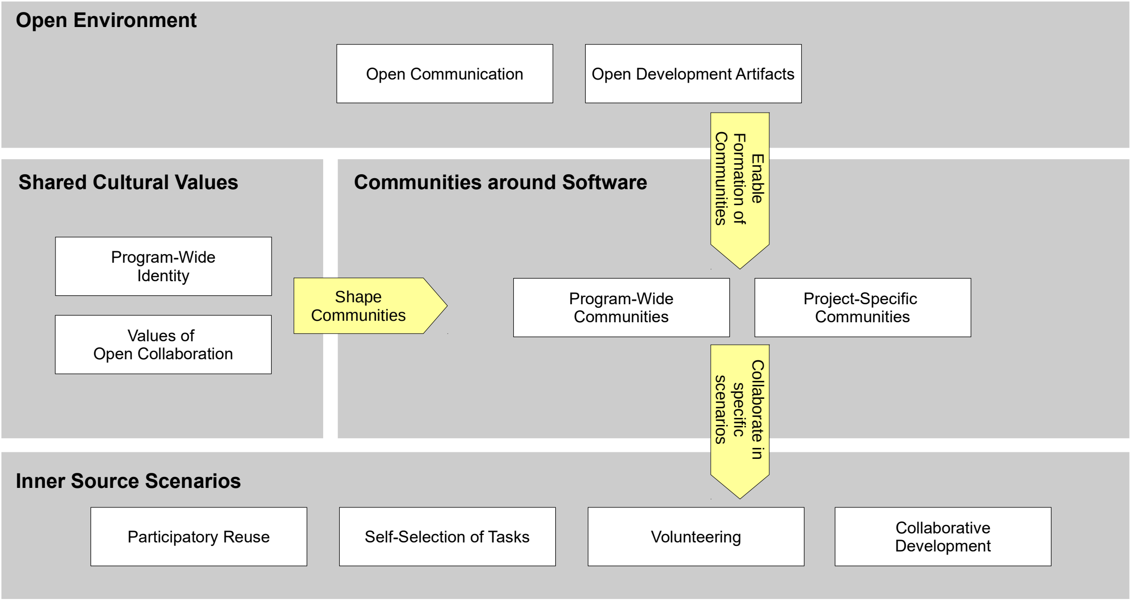
2.3 Key Elements of Inner Source

The discussed definitions define IS and the concepts surrounding it, but they do not describe the elements it is composed of. Therefore, we present a qualitative theoretical model characterizing IS based on four elements that have been extracted from the surveyed literature.

Figure 2.3 shows an overview of our model of elements of IS. The elements in the model are derived from the categories in the code system that resulted from the qualitative data analysis as described in section 2.2. The gray areas are the four elements of IS (top-level categories of our

*MaxQDA is a proprietary software tool that supports the coding process in quliatative data analysis. Further information on MaxQDA can be found on `http://www.maxqda.com`.

Figure 2.3: Theoretical model of inner source elements


Open Environment
Open Communication
Open Development Artifacts
Enable Formation of Communities
Shared Cultural Values
Communities around Software
Program-Wide Identity
Values of Open Collaboration
Shape Communities
Program-Wide Communities
Project-Specific Communities
Collaborate in specific scenarios
Inner Source Scenarios
Participatory Reuse
Self-Selection of Tasks
Volunteering
Collaborative Development

code system regarding the elements of IS). The white boxes (second-level categories) further specify the attributes of these elements. We identified IS to be composed of four key elements:

1. An *open environment* is created by opening up development artifacts, inviting external contributors, and establishing open communication.

2. *Shared cultural values* are internalized by individuals within the organization.

3. Empowered by the open environment and shared cultural values, *communities around software* form.

4. IS collaboration is exercised in specific *IS scenarios* by a project-specific or program-wide community.

The arrows in figure 2.3 indicate the relationships between the elements of IS. The open environment enables developers to form communities around software. The communities around software are shaped by shared cultural values and exercise IS collaboration in specific IS scenarios.

2.3.1 Open Environment

IS embraces an open environment. We found this open environment to be characterized by open communication and open development artifacts. Openness includes the transparency of information and artifacts (Stol et al. 2014) but also the possibility for individuals to participate in projects and communication outside their assigned projects or without a superior's approval (Riehle et al. 2009; Neus and Scherf 2005).

Open communication

In the context of OS, Riehle (2015) defines open communication to be public, written, complete, and archived. The surveyed literature shows that open communication is an element of IS. Neus and Scherf (2005) suggest open communication not only to be public, but also open for everybody to contribute to. Melian (2007) discusses open communication in IS utilizing the framework by Clark and Brennan (1991). She found that open communication impacts all dimensions of communication their framework proposes.

IBM (Vitharana et al. 2010), HP (Melian et al. 2002), Google (Whittaker et al. 2012), Philips (van der Linden 2013), Nokia (Lindman et al. 2010), and DTE Energy (Smith and Garber-Brown 2007) utilize mailing lists to implement open communication. Smith and Garber-Brown (2007) summarizes the benefits of open communication with mailing lists: "[Most community members] preferred to use the mailing lists because the lists allowed them to multitask between receiving help and performing project duties". Martin and Hoffman (2007) summarize that "mailing lists have proven to be an indispensable form of communication between software developers and users". Forums are an alternative implementation of open communication (Microsoft 2008; Lindman et al. 2010; Vitharana et al. 2010). Generally, informal communication channels can be observed in IS (Stol et al. 2014).

However, the surveyed publications do not indicate how complete the open communication was. Contrary to OS, some IS developers may still share a physical working space. We assume this can decrease the completeness of open communication. Smith and Garber-Brown (2007) suggest to focus on open (electronic) communication means whenever possible.

Open development artifacts

IS is characterized by the openness of development artifacts for example, source code or documentation. This openness has two consequences:

First, the open artifacts can be read and reused. IS grants developers "universal access to development artifacts" (Stol et al. 2014). All organizations in the surveyed literature with the exception of Kitware (Martin and Hoffman 2007) opened parts or all of their source code repositories internally. Melian et al. (2002) of HP summarizes: "From the open source development paradigm, [IS] borrows the notion of making source code available freely (openly) for all members of the community". To make source code easily usable as quick as possible, IS, like OS, encourages early and frequent releases of new incremental versions of IS software components (van der Linden 2009; Gurbani et al. 2006; Martin and Lippold 2011; Stol et al. 2014). Internally opened source code is the most obvious but not the only form of open development artifacts (Robbins 2007). Openness of source code alone is not sufficient for enabling collaboration within a community. Also, the information and knowledge regarding IS projects needs to be accessible to the community's members. Van der Linden (2013) concludes that "to support this inner [source] collaboration, the platform documentation should be open. [...] Consequently [...] relevant documentation was published on an internal website, which was easily accessible for all development departments". An important side effect of opening up work artifacts is that not only the artifacts itself but also the work performed on them is publicized. This enables individuals to infer other individuals' and org. units' goals (Dabbish et al. 2012).

Secondly, in addition to reading and using open development artifacts, developers can contribute changes towards code and documentation even if the project is outside of their org. unit's responsibility. Riehle et al. (2009) describe that it was a key design element of the internal forge at the center of their IS program to reduce "the technical and practical hurdle of joining and becoming active in a project" and consequently enhance the projects' openness for contributions. Developers are able to send patches (small packages of code changes) with bug fixes, new features or other additions to the owners of an IS software component (Gurbani et al. 2010; van der Linden 2013). As in OS, the owners review the patch and decide based on its merit whether to reject or accept it (Gurbani et al. 2005, 2006; Riehle et al. 2009)[†]. Once a patch has

[†] Gurbani et al. (2005) is a workshop paper with similar content as Gurbani et al. (2006). We omit the earlier iteration for the remainder of this thesis.

been accepted, it is the IS project's responsibility to maintain this portion of the code. As in OS, individuals can be granted the right to commit patches themselves (so called committers) without going through this review process. At Google, developers that have proven defined skills get write access to a selected subset of the IS portfolio (Whittaker et al. 2012).

2.3.2 Shared Cultural Values

In the IS programs in the surveyed literature, specific cultural values are lived. IS embraces a program-wide identity and the values of open collaboration. While culture is not as easily visible as processes or organizational structure, we found shared cultural values to be an important element of IS.

Program-wide identity

Developers of an organization do not exclusively identify with their organization and its goals. Often developers also identify with their team or with the specific product or component they work on. Developers in IS share a program-wide or even organization-wide identity. They identify with the IS program, the IS projects they are involved in, and the respective IS community.

Such a program-wide identity is desirable from an organization's perspective: The focus of IS communities is not the local success of one individual, team, or org. unit but the success of the whole IS program or even organization (Wesselius 2008; Martin and Lippold 2011). To embrace a program-wide identity, organizations undertook efforts (e.g. trainings and seminars) to establish trust among emplyoees from different org. units (Melian et al. 2002; Martin and Lippold 2011). Melian et al. (2002) summarize that "it is of great importance to establish and build strong working relationships and trust, especially when the work teams are globally distributed".

Values of open collaboration

IS implements the three values of open collaboration as defined by Riehle et al. (2009): egalitarianism, meritocracy, and self-organization.

IS projects are egalitarian. Every contributor who is willing to help an IS project is typically welcome. Contributions to IS projects are typically judged meritocratically based on the value they bring to the project. Meritocracy can also be enabled by open communication as decisions are discussed publicly. To adopt IS, an organization does not necessarily have to become completely self-organizing. Still, IS allows individuals, org. units, and project communities a higher degree of self organization (Riehle et al. 2009).

2.3.3 Communities around Software

Communities around software are a key element of IS. Wesselius (2008) of Philips emphasizes the important role of the community element: "Companies using the [IS software] approach essentially establish an [OS software] community within the confines of their organization." Organizations have implemented program-wide (Dinkelacker et al. 2002; Smith and Garber-Brown 2007; Microsoft 2008) and project-specific IS communities (Melian et al. 2002; Gurbani et al. 2006, 2010; Martin and Hoffman 2007; Riehle et al. 2009; van der Linden 2013).

In OS and IS research, the concept of community has been defined in various ways that are not always compatible to one another and have differing levels of precision. As part of this survey study, we do not aim to resolve this deficiency. For the remainder of this chapter, we define a community broadly as an informal organization of individuals that communicate and collaborate with each other. In IS, these communities cross org. unit boundaries.

Program-wide communities

In IS, not only the project-specific but also the program- or organization-wide view is important. While project-specific communities form around one specific IS project, program-wide communities exist that form around the whole IS program. The participants of this community are a joint set of the participants in project-specific communities. Microsoft (Microsoft 2008) and HP (Dinkelacker et al. 2002) observed the formation of program-wide communities. At DTE Energy (Smith and Garber-Brown 2007) and Kitware (Martin and Hoffman 2007) program-wide communities were stimulated to enable knowledge exchange and networking among individuals within the organization.

Project-specific communities

The main focus of participants in project-specific communities is to collaboratively develop, use or contribute to one specific IS project. Project-specific communities formed around Lucent's SIP transaction manager (Gurbani et al. 2006, 2010), SAP's mobile retail prototype (Riehle et al.

2009), Philips' medical imaging components (Wesselius 2008; van der Linden 2009; van der Linden et al. 2009), and projects at HP (Melian et al. 2002) and Microsoft (Microsoft 2008). Within an IS program, multiple project-specific communities can exist. Project-specific communities can also be found in the OS world (for example, the communities around specific OS projects like JUnit or Libre Office).

2.3.4 Inner Source Scenarios

Based on the surveyed literature, we identified four IS scenarios (participatory reuse, self-selection of tasks, volunteering, collaborative development projects). For a given IS program, it is not necessary that all IS scenarios exist for the program to be called an IS program. We believe that more than the identified IS scenarios can be implemented in IS programs. We propose future research to identify additional scenarios either by observing IS programs or the OS world.

Participatory reuse

Participatory reuse is a form of software reuse in which individuals participate in developing and maintaining the software they reuse (Vitharana et al. 2010). The term was coined by Vitharana et al. (2010) to describe a "scenario in which potential reusers participate in the entire development process (e.g. analysis, design, development, testing) to ensure that the project assets meet their reuse needs". Reuse evolves from a one-way street where existing software is only consumed into a two-way street where developers use and contribute to the software (Wesselius 2008; van der Linden 2013). Individuals can contribute patches to software components they are reusing as part of their work in order to make them fit for their particular needs. Individuals do not self-select which component to contribute to based on their interest or qualification.

Participatory reuse can also be found in the OS world. Wesselius (2008) summarizes: "In [OS], a community works together to develop software. Because the software's users are part of the community, they can add the assets they need."

Self-selection of tasks

Self-selection of tasks allows developers to choose by themselves which development work to perform. To enable self-selection of tasks, Google implemented the "20% time" (Hamel 2008;

Whittaker et al. 2012). The 20% time allows employees to use 20% of their work time to participate on projects outside the scope of their everyday work (Hamel 2008; Whittaker et al. 2012). A Google employee (Google-Blog 2006) reports that "The 20 percent time is a well-known part of our philosophy here, enabling engineers to spend one day a week working on projects that aren't necessarily in our job descriptions." The organization-wide open code repository of Google (Whittaker et al. 2012) enables developers to use this time for contributions to open projects, turning them into IS projects.

Volunteering

Organizations reported about volunteering developers that were motivated to contribute to IS projects in their spare time for fun, to develop their professional skills or to gain reputation (Gurbani et al. 2006; Riehle et al. 2009; Stol et al. 2014). Riehle et al. (2009) summarize: "Even with traditional top-down structured software development organizations, [IS] projects can gather internal volunteer contributions. [...] Volunteers are motivated to contribute, because it is their decision to contribute and they can gain reputation and visibility within the company outside their current primary projects."

Volunteering differs from self-selection of tasks. While in self-selection of tasks developers use working time to perform self-selected programming tasks, volunteers use their spare time for contributing to the organization's IS projects.

Collaborative development projects

The majority of the presented IS scnearios is based on developers contributing code to an IS project via patches. These can either be rejected or accepted by the committers of an IS project.

In OS, it can be observed that a core team of developers creates a component collaboratively. A bazaar-style development as described by Raymond (1999) does not happen instantly but the core team develops the component and shares its ownership. Later, bazaar-style practices, e.g. based on contributing patches, can occur in such a project (Senyard and Michlmayr 2004).

The surveyed literature reported on similar scenarios in IS. Organizations performed collaborative development projects in which they joined resources from different org. units to develop an IS software component these org. units had a shared interest in:

> “GlobalSoft adopted the concept of so-called collaborative development projects. [...] In practice, this resulted in temporary, virtual teams that work together [...] to develop a new (or enrich an existing) component [...].” (Höst et al. 2014) regarding GlobalSoft

> “Our current approach is to start codevelopment [collaborative development] projects in which systems-group and component-group developers work together to create new assets that are relevant to the participating systems group.” (Wesselius 2008) regarding Philips

In collaborative development projects, the involved parties have more influence in the IS projects than contributors in other IS scenarios.

2.4 Classification Framework for Inner Source

We combined findings from the surveyed literature with known IS classification models to derive our classification framework for IS. This classification framework is composed of one classification framework for IS programs and one for IS projects.

We found that IS programs differ in at least three dimensions (prevalence, degree of self-organization, internal economics). Our classification framework for IS programs delivers a multi-dimensional classification of IS programs based on these dimensions. For each of the three dimensions, the framework lays out possible classes. An IS program belongs to exactly one class per dimension (resulting in a total of three classes per IS program). Our classification framework for IS projects works analogously to the classification framework for IS programs. We found IS projects to differ in at least two dimensions (governance, objective).

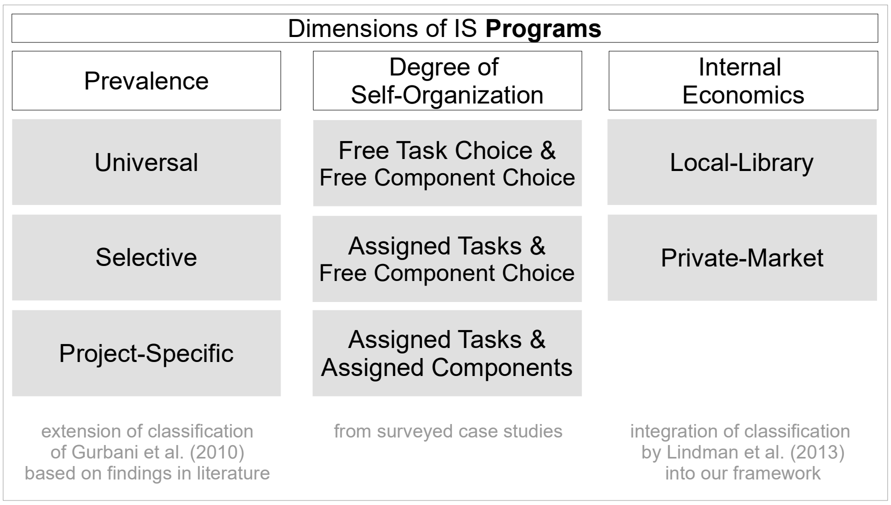
Figure 2.4 and figure 2.4 give an overview of the classification framework, its dimensions, and which classes an IS program or project can fall into for each of the dimensions. Each column represents one dimension. The possible classes are shown as gray boxes. The light gray text under each column indicates how it was derived.

We believe the space of possible IS programs and projects not to be limited to the identified dimensions. We suggest future research to identify additional dimensions or classes of IS programs and projects.

2.4.1 Classification of Inner Source Programs

The classification framework of IS **programs** is based on the three dimensions *prevalence*, *degree of self-organization*, and *internal economics*. The prevalence dimension is based on the

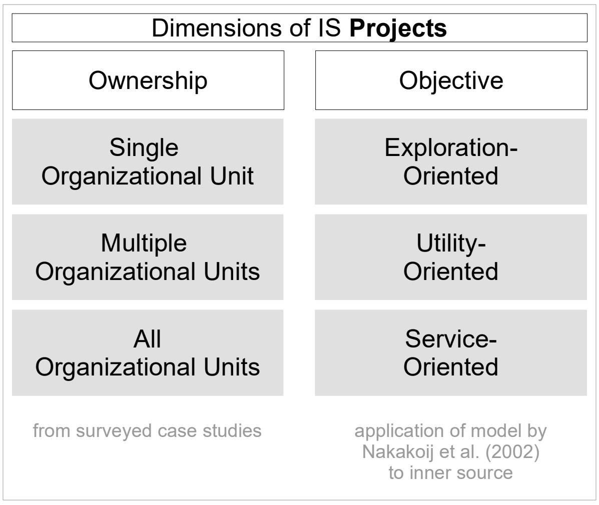
Figure 2.4: Classification framework of inner source programs



Dimensions of IS **Programs**

Prevalence	Degree of Self-Organization	Internal Economics
Universal	Free Task Choice & Free Component Choice	Local-Library
Selective	Assigned Tasks & Free Component Choice	Private-Market
Project-Specific	Assigned Tasks & Assigned Components	
extension of classification of Gurbani et al. (2010) based on findings in literature	from surveyed case studies	integration of classification by Lindman et al. (2013) into our framework

Figure 2.5: Classification framework of inner source projects



Dimensions of IS **Projects**

Ownership	Objective
Single Organizational Unit	Exploration-Oriented
Multiple Organizational Units	Utility-Oriented
All Organizational Units	Service-Oriented
from surveyed case studies	application of model by Nakakoij et al. (2002) to inner source

classification model presented by Gurbani et al. (2010). As a result of our qualitative data analysis of the surveyed literature (primarily the surveyed case studies), we extended their model to form the prevalence dimension. The internal economics dimension integrated the classification model of Lindman et al. (2013) into our framework. The different degrees of self-organization were derived solely from qualitative data analysis. The next paragraphs discuss these three dimensions in more detail.

Prevalence

Gurbani et al. (2010) presented a classification of IS programs based on their prevalence within the organization. They distinguish project-specific and infrastructure-based IS programs. Project-specific IS programs are focused on one specific IS project which is usually a primary technology of the organization, of high strategic or operative importance, or has many stakeholders relying on it. In a project-specific IS program the IS project provides the development infrastructure. Contrary to this, in infrastructure-based IS programs, the organization provides development infrastructure and enables individuals or org. units to host their IS projects on it. In an infrastructure-based IS program, IS has a higher prevalence within the organization (Gurbani et al. 2010).

In the surveyed literature, we found that infrastructure-based IS programs can be differentiated further. A fraction of the infrastructure-based IS programs inner-sourced only some components while in others all of the organization's software components were inner-sourced. We call these IS programs *selective* and *universal* IS programs. Consequently, we extend the classification by Gurbani et al. (2010) and consider the following three classes of an IS program's prevalence:

- *Universal*: All of the organization's software artifacts are publicized as an IS software component. There is no software component that is not inner-sourced.

- *Selective*: Only selected software artifacts are publicized as IS software. The remaining software components are not inner-sourced. Consequently, many IS projects exist.

- *Project-Specific* (Gurbani et al. 2010): The majority of software components is not inner-sourced. Only a specific IS project is run within the IS program.

Of the three presented variants, implementing universal IS program has the largest impact on the organization: Every software component is made internally available.

An organization running a universal IS program may decide to exclude a few components from the IS portfolio for idiosyncratic reasons (for example, due to security or intellectual property concerns). We suggest to still classify such IS programs as universal IS if the vast majority of software is IS software and components are only excluded from the IS portfolio on strictly exceptional basis.

Degree of Self-Organization

The IS programs described by the surveyed literature grant individuals a varying degree of self-organization by allowing or not allowing them to self-responsibly choose which IS software components to reuse and/or which tasks within the IS program to work on. In the surveyed literature, we found the following three classes:

- *Free task choice and free component choice*: Individuals can choose which components to reuse and (at least a fraction of) their every-day tasks. Consequently, they not only contribute to IS software components used as part of their assigned everyday work. They are also enabled to contribute to IS software components that match their personal interests or expertise.

- *Assigned Tasks and free component choice*: Tasks are assigned to the individual developer in a traditional way. However, individuals can choose which IS software components to reuse for finishing their assigned tasks. No corset forces an individual to reuse one specific IS software component.

- *Assigned Tasks and Assigned Components*: The elements of IS (see section 2.3) are implemented. However, no other freedoms are granted to individuals. They have no right to freely choose which IS software components to reuse as part of their work. The used components are predefined by an existing corset (e.g. a software product line (SPL) setup).

We did not find any indication of a fourth class *free task choice and assigned components* in the surveyed literature.

Internal Economics

We integrate the classification of IS programs by Lindman et al. (2013) into our classification framework. They classify IS programs based on the IS program's internal economics into the following two classes:

- *Local-library*: Every party within the program-wide community can reuse the IS software asset free of charge. Contributions to the IS portfolio are not reimbursed or specially expedited (Lindman et al. 2013).
- *Private-market*: Internal market mechanisms are in place to regulate and steer contributions and reuse (Lindman et al. 2013).

One could argue that an IS program with a private-market is not an IS program at all, as it potentially hinders reuse and collaboration. However, in our experience some organizations implement complex cost allocation schemes which make it necessary to use private-market IS programs. Lindman et al. (2013) summarizes that a private-market "does present some benefits of open innovation (ideas flowing freely, quick diffusion of inventions to enable incremental innovation, reuse) while addressing the appropriation in a fairly practical manner".

2.4.2 Classification of Inner Source Projects

We classify IS **projects** based on two dimensions: The governance dimension describes who is responsible for the project and the developed IS software component. The objective dimension describes what the project is aiming to achieve. Our qualitative data analysis of the surveyed literature (primarily the surveyed case studies) indicated IS projects' variations in both dimensions. For the governance dimension, the classes result exclusively from our qualitative data analysis. For the objective dimension, we integrated a project classification model from the OS world (Nakakoji et al. 2002) into our framework.

Governance

Governance of IS projects and regard IS software components was implemented in different ways by the IS projects described in the literature. As a result of our qualitative data analysis, we identified three classes that describe how governance of IS projects was implemented. We observed governance by a *single org. unit*, *multiple org. units*, and *all org. units*.

- *Single org. unit*: The IS project is explicitly governed by one single org. unit.
- *Multiple org. units (governance board)*: The IS project is governed by a board formed of multiple org. units.
- *All org. units*: The IS project is not governed by a select group of org. units. The IS software component is seen as commodity. Governance and ownership is shared between all org. units in the organization.

Both governance by a single or multiple org. units require an explicit proclamation of responsible org. units for the outcomes of an IS project. We believe collaborative development projects as described in section 2.3 to typically result in governance of an IS project being executed by multiple org. units because in such projects multiple parties have a stake in the IS project from the very beginning.

Gurbani et al. (2010) reported on roles they implemented to enable the govenrance of a IS project governed by multiple org. units. They defined explicit management roles (project managers, software architects) and roles for mediation between the core team of the IS project and its users.

In Stellman and Greene (2009), Auke Jilderda describes that explicitly defined ownership (and consequently governance by these owners) is an important attribute of each IS software component. He argues that some entity should always have a final say about the development direction or on whether to accept a contribution. However, Whittaker et al. (2012) and Linåker et al. (2014) discussed IS projects that were not governed by select org. units but by all org. units equally. Whittaker et al. (2012) uses the term "shared" components or libraries to refer to IS software components resulting from such projects. The surveyed literature did not present examples of IS projects governed by all org. units. However, the literature discussed the characteristics of such projects.

Linåker et al. (2014) discuss that such IS projects "do not need any administration or anyone responsible for the development of the component".

At Google, multiple of such projects exist (Whittaker et al. 2012). While the governance of these projects is independent of particular org. units, a developer has to follow defined rules when contributing to these projects (Whittaker et al. 2012). Before performing a modification, a committee needs to certify the developer's proficiency in the relevant programming languages

(Whittaker et al. 2012). Strict requirements to the test coverage of shared components and a mandatory review process are in place to mitigate quality degregation (Whittaker et al. 2012). Also, a developer is responsible for adapting other components that depend on the modified component if necessary (Whittaker et al. 2012).

Still, IS projects governed by all org. units equally are not broadly researched. It is unclear how conflicts can be resolved effectively, which components are fit to be governed in such a way, or how software evolution can be managed in such projects. We suggest future research to address these issues.

Objective

IS projects described in literature served different objectives. In the OS context, Nakakoji et al. (2002) identified three different classes of projects depending on the project's primary objectives. Based on the surveyed literature, we found their objectives to be fit for classifying IS projects as well. In analogy to Nakakoji et al. (2002), we use the following three classes to express the objectives of an IS project:

- *Exploration-oriented*: The IS project aims to make innovation accesible to the whole program-wide IS community. Nakakoji et al. (2002) note that due to their "epistemic nature" such projects usually have high quality requirements. Contribution of feedback (e.g. via mailing lists) is particularly important for an exploration-oriented project.

- *Utility-oriented*: The IS project aims to fill an immediate need in functionality. Typically, the developers of the initial code are an individual or a small party who "cannot find an existing program that fulfills their needs completely" (Nakakoji et al. 2002). Utility-oriented projects usually have only a small project-specific community or their community exists as part of larger community (e.g. if the utility-oriented project is part of the ecosystem of another IS project).

- *Service-oriented*: The IS project's main goal is to provide "stable and robust services" to end-users of the IS software software (Nakakoji et al. 2002). Service-oriented projects typically produce business critical IS software software components, have high quality requirements, and are conservative against rapid changes (Nakakoji et al. 2002).

An OS project's objective can change during its life-cycle (Nakakoji et al. 2002). As in OS, an IS project can evolve and its objective change. Lucent's SIP transaction manager (Gurbani et al. 2006, 2010) started as an exploration-oriented IS project. Its aim was to make innovation (in the form of the implementation of a new telephony protocol) available within the organization. Later, it evolved into a service-oriented IS project as many products of Lucent started to rely on it (Gurbani et al. 2006).

2.5 Application of the Classification Framework

In this section, we apply the classification framework to the known instances of IS programs and projects. The presented classification framework was derived from the IS programs and projects in surveyed literature. Consequently, the application of the model to the same set of programs and projects cannot serve as validation. Rather, the application of the framework serves as a demonstration of its capabilities.

2.5.1 Application to Inner Source Programs

We classified 12 of the IS programs from the surveyed literature according to the three dimensions of our classification framework for IS programs.

Table 2.2 presents how each of the known IS programs is classified according to our classification framework for IS programs. The different classes of the *prevalence* dimension are expressed as columns; the different classes of the *degree of self-organization* dimension are expressed as rows.

Dimension: Internal economics

Table 2.2 does not show the market-mechanisms dimension because we did not find an implementation of a private-market IS program. However, the private-market idea was discussed in the context of Nokia (Lindman et al. 2013) and Philips (Wesselius 2008; Lindman et al. 2013). The roots of Philips' IS program show elements of private-market IS. Initially, org. units were required to pay a fee for reusing a component (Wesselius 2008). However, contributions from outsiders neither occurred nor were they reimbursed to form an private market.

Table 2.2: Classification of inner source programs

		Prevalence		
		Project-Specific	Selective	Universal
Degree of Self-Organization	Assigned Tasks & Assigned Comp.	• GlobalSoft / SoftCom • Philips	none	none
	Assigned Tasks & Free Comp. Choice	• Lucent	• HP (CDP) • HP (CS) • IBM (CMS • IBM (IIOSB) • Microsoft (CodeBox) • Nokia • SAP	• DTE Energy
	Free Task Choice & Free Comp. Choice	none	none	• Google

Dimension: Degree of Self-Organization

Regarding the degree of self-organization, we observed IS programs with assigned tasks and components (2), assigned tasks and free component choice (9), and free task and component choice (1). IS programs with assigned tasks and components can be found at Philips (van der Linden 2009; van der Linden et al. 2009; van der Linden 2013) and GlobalSoft (Höst et al. 2014) who use IS in the context of their SPL development. IS programs with free component choice but assigned tasks were implemented at SAP (Riehle et al. 2009), Microsoft (2008), and other organizations. Google implemented free choice of components and (to a certain degree) tasks. They allow developers to use 20% of their time for performing work they are interested in or deem necessary (Whittaker et al. 2012; Hamel 2008). While Google employees may use it to initiate a new project, it is also used for contributing to IS projects (Google-Blog 2006).

For the Department of Defense's Forge.mil program (Martin and Lippold 2011) we were not able to determine the degree of self-organization or their internal economics.

Dimension: Prevalence

In the prevalence dimension, we observed selective (8), project-specific (3), and universal (2) IS programs. Google's shared source code repository (Whittaker et al. 2012) is an example of a universal IS program. Also, DTE Energy reported to have opened up their repositories to enable "all developers to see all code across the enterprise" (Smith and Garber-Brown 2007). SAP's internal software forge (Riehle et al. 2009) and IBM's community source (Vitharana et al. 2010) are examples for selective IS programs. Only selected components are developed using IS. Nokia's iSource program is an instance of a selective IS program. As iSource is the standard platform for all new projects that use a specific source code management (SCM) system (Lindman et al. 2010, 2013), it could become a universal IS program gradually.

Discussion

Table 2.2 shows that a majority of the surveyed organizations implemented selective (8), and local-library (10) IS programs with assigned tasks and free component choice (9). IS programs with assigned tasks and components (2) or free task choice (1) as well as universal IS programs (2) have a low prevalence.

Despite the few instances of programs with assigned tasks and components, we believe these IS programs to be of high relevance for industry. Many organizations develop software in org. units that are in a static relationship to other org. units and the software they develop (for example due to SPL engineering). At least one organization (Siemens) that implemented software SPL engineering indicated that IS can benefit their development (Bartholdt and Becker 2012).

Two out of the three project-specific IS programs also had assigned tasks and components. In the cases of Philips (van der Linden 2013) and GlobalSoft (Stol 2011; Höst et al. 2014), these IS programs were used to augment SPL development and mitigate challenges in requirements elicitation, feature prioritization, and bottlenecks in the platform org. units.

However, IS programs with free component choice can be project-specific as well. Lucent only developed one IS software component. Still, there are no reports that this component was imposed on the developers. Org. units were able to decide for themselves whether or not to use the IS software component (Gurbani et al. 2006, 2010).

Based on the information in the surveyed literature, we were only able to classify a small amount IS programs (12). This leads to two problems:

- We were not able to identify IS programs for all possible combinations of classes. Consequently, table 2.2 contains empty cells. However, we did find no indication that the combinations with no example IS program are invalid.

- The small sample size does not allow us to draw conclusions about correlations between classes of IS programs. We suspect that more self-organization and a higher prevalence of an IS program could be correlated. A potential cause is that organizations that take the risk of rolling out IS on a broader scope (higher prevalence) are also more aware of the potential benefits of a higher degree of self-organization. For the surveyed IS programs, the prevalence and degree of self-organization dimension show higher diversity. Regarding the internal economics dimension we observed less diversity.

We suggest further research to identify and classify additional IS programs and explore correlations between classes of IS programs in the different dimensions.

Table 2.3: Classification of inner source projects

		Objective		
		Exploration-Oriented	Utility-Oriented	Service-Oriented
Governance	Single Organizational Unit	• Lucent (SIP Server, earlier) • SAP (Mobile Retail Demo)	• Microsoft (PEX)	• Microsoft (CodeBox)
	Multiple Organizational Units	none	none	• GlobalSoft (No name) • Lucent (SIP Server, later) • Philips (Medical Image SPL)
	All Organizational Units	none	none	none

2.5.2 Application to Inner Source Projects

Six IS projects in the survyed literature were described with sufficient detail to allow us to apply our framework for classifying IS projects. For one of these projects, Lucent's SIP server (Gurbani et al. 2010), literature reported two different phases which we consider seperately. For the other nine IS programs, literature did not report about specific projects. Table 2.3 summarizes the classification of the six considered IS projects. The table follows the same logic as the previous table 2.2.

Dimension: Governance

Regarding the governance dimension, literature reported projects governed by a single org. unit (4) and multiple org. units (2). A specific project without a governing org. unit was not presented in literature. However, such projects were implemented at Google (Whittaker et al. 2012) and discussed in the context of the case study by Linåker et al. (2014). We suggest further exploratory research on IS projects governed by all org. units and the implications for software quality and governance.

Dimension: Objective

Regarding the objectives, the surveyed literature reported service-oriented (4), exploration-oriented (2), and utility-oriented (1) projects. Nakakoji et al. (2002) discussed that exploration- and utility-oriented OS projects are typically governed by a single org. unit and evolve into service-oriented projects owned by multiple org. units. This could be true for IS projects as well and we suggest future research: Lucent's SIP server changed from being governed by one to being governed by multiple org. units after becoming a service-oriented IS project. Microsoft's Code Box project discussed the need to empower the contributors and users of the service-oriented project further (Microsoft 2008). The service-oriented project at GlobalSoft implemented a steering committee (Stol 2011; Höst et al. 2014).

Discussion

Based on the information in the surveyed literature, we were only able to classify a small amount of IS projects (6). Consequently, table 2.3 contains empty cells because we were not able to identify example IS projects for every possible combination of IS classes. We found no indiciation in the surveyed literature that specific combinations if IS projects classes are invalid.

In the surveyed IS projects, we observed a correlation between the class of an IS projects and the IS scenarios that are common in the IS program hosting it: Volunteering or self-selection of tasks were observed in all IS programs running exploration-oriented IS projects. We assume innovative exploration-oriented projects to have a higher potential to attract interested individuals. Project-specific IS programs often ran service oriented projects (3/4). Gurbani et al. (2010) describe that focusing IS efforts on one specific project is a beneficial way to deal with critical assets. All IS programs in the surveyed literature with assigned tasks and components implemented service-oriented projects.

2.6 Conclusion

The majority of scientific publications regarding IS describe the phenomenon in the context of one or a few organizations. However, the research area lacked a systematic arrangement of prior research results. With the literature survey presented in this chapter, we systematically arranged prior research results to resolve this shortcoming.

We provided a holistic definition of IS and related concepts, a model of key elements constituting IS, and the first classification framework for IS programs and projects. We applied our classification framework to demonstrate its capabilities and delivered a map of already known IS programs and projects. While there are four key elements that constitute IS (values of open collaboration, an open development environment, communities around software, specific IS scenarios), IS programs and projects differ from one another. Researchers can use our model of IS elements as a lens to discuss the IS programs they study and the classification framework to reason about the generalizability of IS related theories.

3

Patch-Flow Measurement Method

The majority of scientific literature regarding inner source (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 magnitude of IS and participating parties in it. We are not aware of any established method that can be used to measure or quantify 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 chapter, we present a method to quantify IS collaboration by measuring an organization's patch-flow. In open source (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 (org. 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.

This chapter answers the following research question:

- RQ3: 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 chapter 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 chapter is structured as follows. Section 3.1 gives an overview of the related work detailing prior work regarding IS and measuring software development collaboration. Section 3.2 introduces the patch-flow measurement method by defining the contribution-flow and patch-flow phenomena, data structures to represent patch-flow and a process to measure it. We evaluate the patch-flow measurement method using case study research. Section 3.3 describes our evaluation approach. Section 3.4 presents the findings of our evaluating case study and section 3.5 a discussion of these findings. Section 3.6 discusses the trustworthiness of our study. Section 3.7 closes the chapter with a conclusion.

3.1 Related Work

We discuss how related work has measured collaboration in the context of IS and then discuss how prior work measured collaboration in the broader context of software development.

3.1.1 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 (Smith and Garber-Brown

2007), Ericsson (Torkar et al. 2011), Hewlett-Packard (Dinkelacker et al. 2002; Melian and Mähring 2008), IBM (Vitharana et al. 2010), Kitware (Martin and Hoffman 2007), Lucent (Gurbani et al. 2006, 2010), Nokia (Lindman et al. 2008), Paypal (Oram 2015), Philips (van der Linden et al. 2009), Rolls-Royce (Stol et al. 2014), and SAP (Riehle et al. 2009).

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 (Dinkelacker et al. 2002; Riehle et al. 2009). Other organizations extend these metrics by counting the number of developers contributing to or being committers in each IS project (Lindman et al. 2008; Torkar et al. 2011). In contrast to our work, they do not discuss how actively IS projects receive IS contributions. Gurbani et al. (2006) 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. (2010) 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.

3.1.2 Measuring Software Development Collaboration

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

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" (Meneely and Williams 2011). Previous research utilizes primarily two types of social networks (Meneely and Williams 2011): Developers networks (Madey et al. 2002; Crowston and Howison 2005; Schwind and Wegmann 2008; Costa et al. 2014; Tymchuk et al. 2014; Joblin et al. 2015) and contribution networks (Pinzger et al. 2008).

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 (SCM) systems, communication logs, issue tracking systems, or other sources (Meneely and Williams 2011). Developer networks are capable of representing relationships among developers (Meneely and Williams 2011) and model the structure of development communities (Joblin et al. 2015). In contrast, this chapter presents patch-flow graphs that use the structure of a given organization and model to what magnitude parties within this structure (typically org. units) contribute to one another.

Contribution networks are graphs showing contributions of developers to one or many source code files (Pinzger et al. 2008; Meneely and Williams 2011). 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 (org. 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.

Other Approaches

Gousios et al. (2008) and Kalliamvakou et al. (2009) present a taxonomy to measure and classify contributions to OS 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 OS 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. (2010) 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.

3.2 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.

3.2.1 Contribution-Flow Phenomenon

IS practices can lead to a phenomenon we call contribution-flow. We define the following concepts:

- A *contribution* to an IS project is the result of any work performed on that project.
- An *IS contribution* is any contribution made to an IS project by a developer who is external to an IS project.
- *Contribution-flow* is the flow of IS contributions across organizational boundaries such as project or org. unit boundaries within an organization.

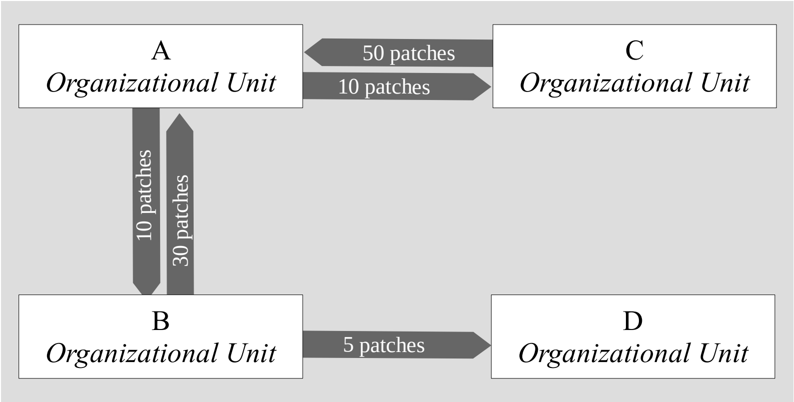
A developer is considered external to a project if not a member of the org. unit owning the IS project. A contribution can be a code or non-code contribution. Non-code contributions include the raising of issues in a IS project's issue tracker, the contribution to a discussion on a mailing list, providing code review comments for a code contribution, and others.

3.2.2 Patch-Flow Phenomenon

Patch-flow is a special case of contribution-flow. 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 an IS project.
- *Patch-flow* is the flow of patches across organizational boundaries such as project or org. unit boundaries within an organization.

Figure 3.1: Example patch-flow between four organizational units



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 (Gurbani et al. 2006; Riehle et al. 2009).

Example

Figure 3.1 displays example patch-flow involving four org. units A, B, C, D. The white boxes are org. units; the gray arrows indicate patch-flow between two org. units.

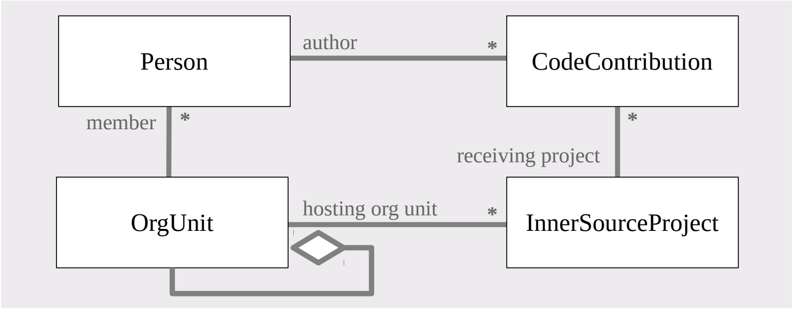
In the example, ten patches flow from org. unit A to B. That means developers allocated to work for org. unit A contributed ten patches to components that are owned by org. unit B. A total amount of 80 patches is received by A that only contributed 20 patches to B and C in total.

Relationship to Inner Source

An individual within an organization takes part in IS collaboration by contributing to an IS project provided by another org. 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 the remainder of this thesis, we do not consider the more generic contribution-flow but focus on the flow of code contributions. This is not to discredit non-code contributions which themselves can be of high value for an IS project and an organization. However, because software code is the primary artifact resulting from a software development effort and a majority

Figure 3.2: Simplified patch-flow flow data model as UML2 class diagram



of other contributions will eventually result in code contributions, the patch-flow is an appropriate measure for IS collaboration. More patch-flow indicates more IS collaboration.

3.2.3 Data Structures

In this section, 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.

Object-Oriented Patch-Flow Model

Figure 3.2 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 org. unit (OrgUnit class). Org. units are modeled using the composite design pattern (Gamma et al. 1994): An org. unit can be composed of child org. units.

Patch-Flow Graph

We define the term patch-flow graph as follows:

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

Each edge weight represents the number of patches flowing.

A patch-flow graph has a density. We define patch-flow density as follows:

- The *density D* of a patch-flow graph (or any directed graph) is defined as $D = E/N(N-1)$ with N being the number of nodes (org. units) and E being the number of edges (patch-flow relationships) in the graph.

The density of a patch-flow graph describes how many of the possible patch-flow relationships exist.

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 org. units. To distinguish patch-flow between org. 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 org. units have level 1. Their children org. units have level 2 etc. Level 0 is considered the highest level. We define the following terms:

- The *lowest common ancestor* (LCA) of two org. 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 LCA of the contributing and the receiving org. unit.

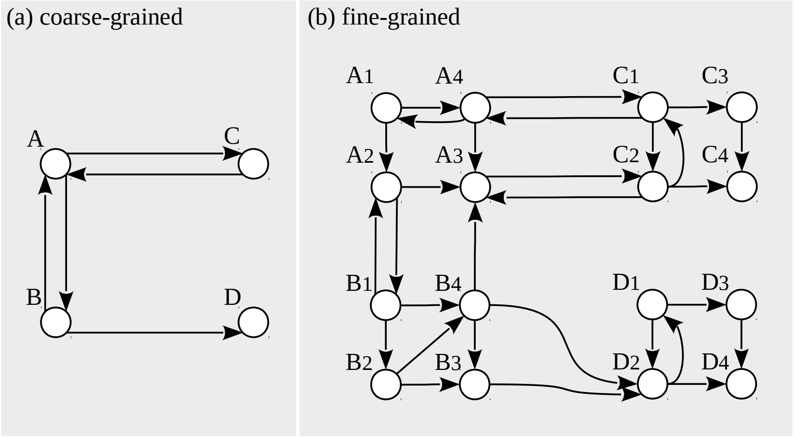
Code contributions between descendants and ancestors not considered patch-flow. Patch-flow crossing level n always crosses level $n+1$. The *highest level crossed* of a patch-flow ($n = n_{lca} - 1$) serves as a metric for the distance between the two involved org. units: Two teams in to separate coarse-grained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

Figure 3.3 shows two patch-flow graphs visualizing patch-flow in the example organization from the previous section. The organization is composed of org. 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 org. 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).

Figure 3.3: Patch-flow graphs with different granularity



Constructing different patch-flow graphs allows researchers and practitioners to study the patch-flow between org. 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.

3.2.4 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 steps:

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 org. unit.* Map the authors of a code contribution to their org. unit.

6. *Mapping of IS projects to org. units.* Map the IS projects to the org. units responsible for them. For some IS projects, it might not be possible to allocate an org. 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 org. units and authors are already identified. Ideally, all activities are automated to reduce measurement costs and risk of human errors during measurement.

3.2.5 Relationship to Classification

This section discusses how the patch-flow method can be applied to the different classes of IS projects and programs introduced in chapter 2.

IS programs. None of the discussed dimensions on which IS programs differ from one another (degree of self-organization, prevalence, internal economics) have an immediate impact on the six steps for patch-flow measurement presented above. The patch-flow in the context of an IS program can be measured independently of its class along all three discussed dimensions.

IS PROJECTS. The objective of an IS project is irrelevant for performing the six steps of the patch-flow measurement methods. Patch-flow can be measured for exploration-oriented, service-oriented, and utility-oriented IS projects.

However, the governance dimension is relevant for patch-flow measurement: Step 6 of the patch-flow measurement method is to map each IS projects to the org. units that is responsible for it.

In chapter 2, we established the IS projects can be governed by one org. unit, multiple org. units, or all org. units of an organization. Typically, this translates into one, multiple, or all org. units considering themselves responsible for the IS projects.

How can the patch-flow be measured for IS projects that are owned by multiple or all org. units of an organization if the patch-flow method requires each IS project to be mapped to one responsible org. unit? In such scenarios, multiple strategies allow to still assign a responsible org. unit and thus measure the patch-flow:

1. *Assign IS project to virtual org. unit.* One strategy is to assign the IS project to a virtual IS org. unit that does not exist in the real organizational structure. As a consequence, all code contributions to the IS project, will be considered patch-flow into the virtual IS org. unit. Potentially, this org. unit can host multiple project.

2. *Assign IS project to org. unit that contributes most.* Another strategy is to still assign the IS project to one and only on org. unit. Researchers and practitioners can define arbitrary criteria on how to perform such mapping. A sensible approach is to assign the IS project to the org. unit that performed the majority of code contributions in the considered time interval.

3. *Assign IS project to multiple org. units.* If required, one can consider adapting the patch-flow method (particularly step 6) and the patch-flow data model (see section 3.2.3) to deal with an $n : m$ mapping between IS projects and org. units. One would then only interpret code contributions as patch-flow if they are not by one of the org. units responsible for the IS project.

Strategy (1) is easy to implement. However, it can also lead to wrong interpretations of the collected data: If only one IS project contributes to an IS project and this project is assigned to the virtual org. unit, an individual looking at the patch-flow might wrongly assume that said IS project attracts a large number outside contributions. Strategy (2) mitigates this problem and does not lead to an over-representation of patch-flow in an organization. Strategy (3) in itself can lead to wrong interpretations too because the collaboration between the multiple responsible org. units is not included in patch-flow (despite it being an instance of IS collaboration). Which strategy to choose depends on the specific research question or analysis objective at hand.

3.3 Evaluation Approach

We evaluated the patch-flow measurement method by applying it in an industry case study following Yin (2013). Our case study is a *single-case* case study (Yin 2013) because we studied one organizations with one isolated context and a *hollistic* case study (Yin 2013) because we did not consider multiple units of analysis.

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.

3.3.1 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. The organization brought us in as consultants and, by doing so, partially financed this research. Upon request of our partners in the case organization, we do not disclose the real name of the organization[*].

[*] The organization is also part of the multiple-case case study presented in section 5. There we refer to the organization as *industry org.*

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.

3.3.2 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.

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 3.1 presents details about the interviews and workshop. We reference each interview and workshop throughout this chapter 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.

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 3.2.4). We considered code contributions

Table 3.1: 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

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 org. units of each developer (i.e. business units), they did not contain the lower level org. 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 org. 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 org. unit (activity 5), and using an internal database assigned the owning org. unit (activity 6). Like Guzzi et al. (2012), 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 org. 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.

3.4 Evaluation Results

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

3.4.1 Organizational Structure

Figure 3.4 displays an excerpt of the organizational structure of the studied organization. The nodes (circles) represent org. units. The edges (lines) indicate child-parent relationships between org. units. The nodes are annotated with letters that are used to construct org. unit identifiers (if a node is annotated with *b* and its parent node with *a*, then the identifier is *ab*). The excerpt only contains org. 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 org. units on each level (interview *I*2). Using this type information, we adjusted the level for *ab*, *baba*, and their descendants. We assigned each organizational level (org. level) one color that we use to identify the level throughout this chapter.

We refer to the most coarse-grained org. 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 *I*1, *I*2). 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. Org. 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 org. units and typically consist of 5 to 15 developers. Each team has a specific technical task regarding

Figure 3.4: Organizational structure including all organizational units articipating in inner source collaboration

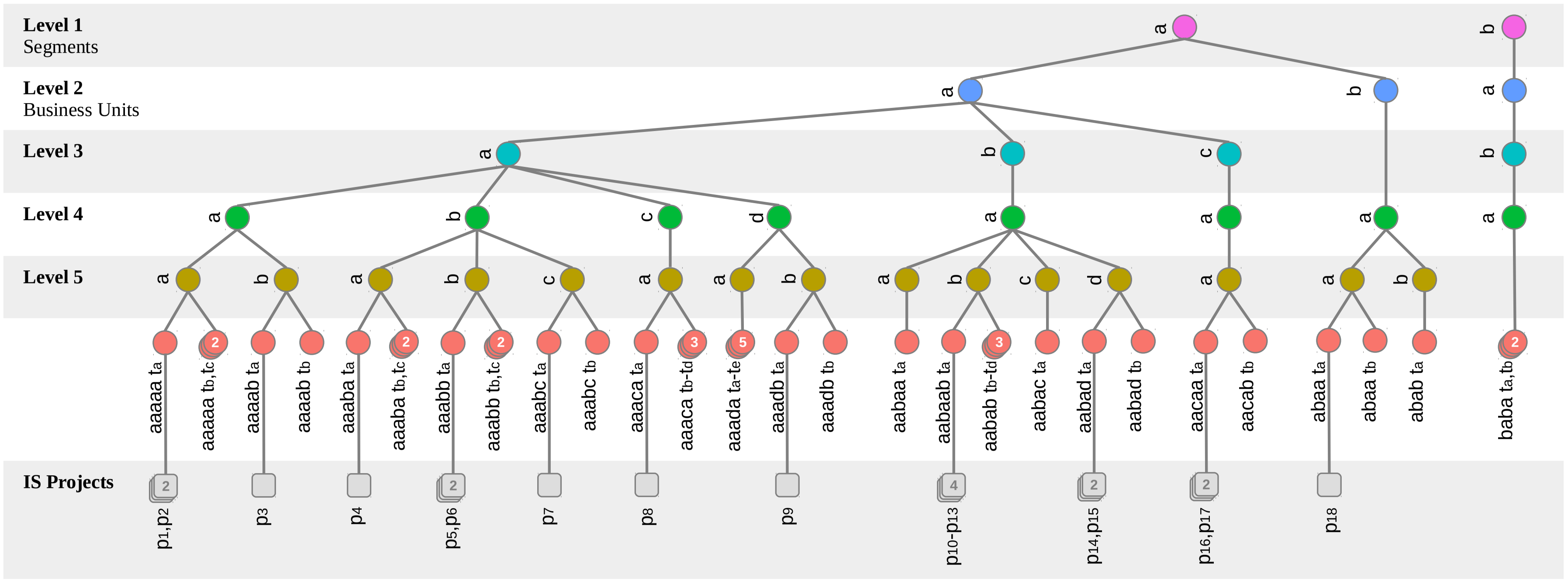
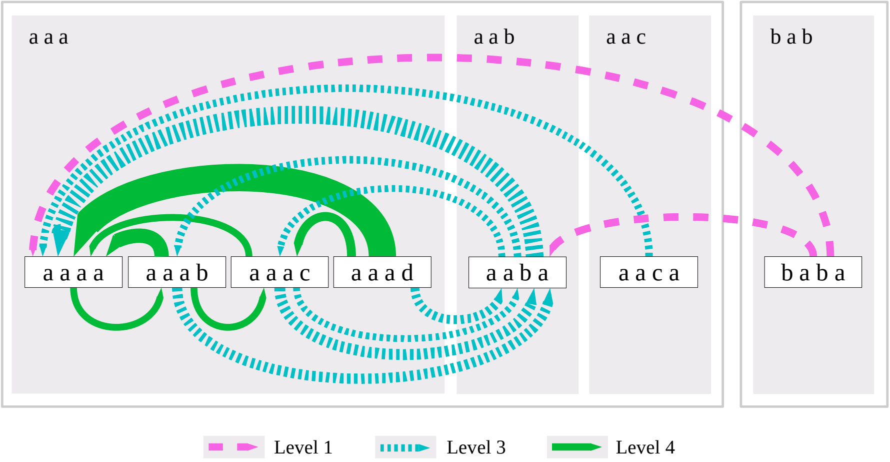


Figure 3.5: Patch-flow graph showing patch-flow between the level 4 organizational units



one part of a product. Figure 3.4 displays each team with its full identifier consisting of the parent org. unit (e.g. *aaaaa*) and a team identifier (t_i, with index i of a, b, c, ...).

Eleven teams own one or more IS projects (gray boxes in figure 3.4). 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 org. units could own an IS project or employees assigned to a higher level org. unit could decide to contribute patches.

3.4.2 Patch-Flow Overview

Figure 3.5 shows the patch-flow aggregated by level 4 org. units as a graph. For comprehensibility, we use a different notation than for the previous patch-flow graphs: The white boxes represent level 4 org. units. The figure also displays a part of the organizational hierarchy. Level 4 org. units are contained by gray boxes representing level 3 org. units. Gray lines around these boxes indicate level 2 org. units (business units). Directed edges between the level 4 org. units represent the patch-flow.

The width of an edge shows its weight (the amount of patches flowing). The edges spin mathematically positive (counter clock-wise). 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 org. units contribute or receive patches crossing level 4 and are consequently included in figure 3.5. Three of them (*aaad*, *aaca*, *baba*) do not receive patches. The org. units *aaad* and *aaca* receive no patches crossing level 4 despite hosting IS projects. The org. 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 org. units of other segments.

Identifying contribution activity

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

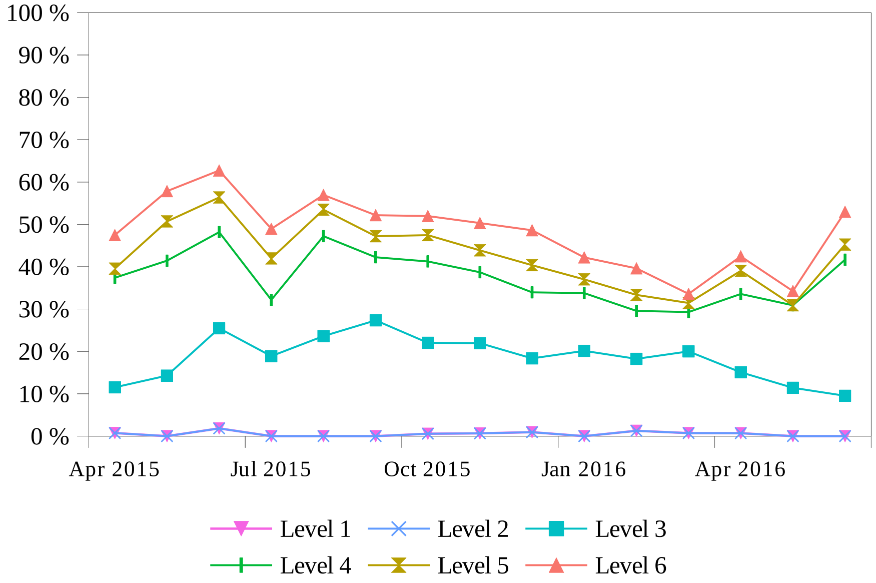
Identifying contribution receival

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

Identifying collaboration relationships

An org. unit might collaborate only with select org. units. For example, we observed intense collaboration between the children org. units of *aaa* and *aab* compared to other level three org. units. Children of *aaa* contribute 131 patches to *aab*; children of *aab* contribute 172 patches

Figure 3.6: Patch-flow relative to all code contributions over time by highest level crossed



to *aaa*. The patch-flow graph is capable of expressing how intense two org. units collaborate with one another (sum of edge weights between these two org. units).

3.4.3 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 org. 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 org. units (e.g. teams) can have a different meaning than a patch-flow between two top level org. 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 3.6 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 org. units (teams). The yellow line shows the percentage of patches flowing between level 5 org. 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 org. units:

- 18.5% across level 3

We observe nearly absent collaboration (insignificant patch-flow) among level 2 and 1 org. 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 org. 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 IS practices they can still be considered so called "silos" (org. units that do not collaborate).

Identifying silo boundaries

A visualization of the patch-flow crossing different org. levels as in figure 3.6 is capable to express on what org. levels silos have formed and across which levels how much IS collaboration happens.

3.4.4 Patch-Flow into IS Projects

Figure 3.7 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.

Figure 3.7: Absolute number of code contributions received by inner source project

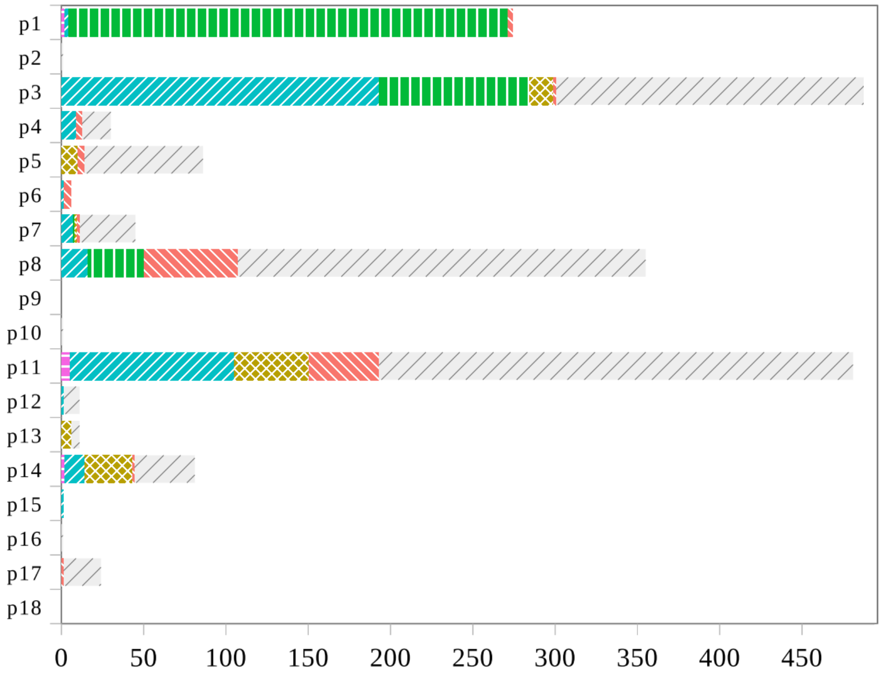
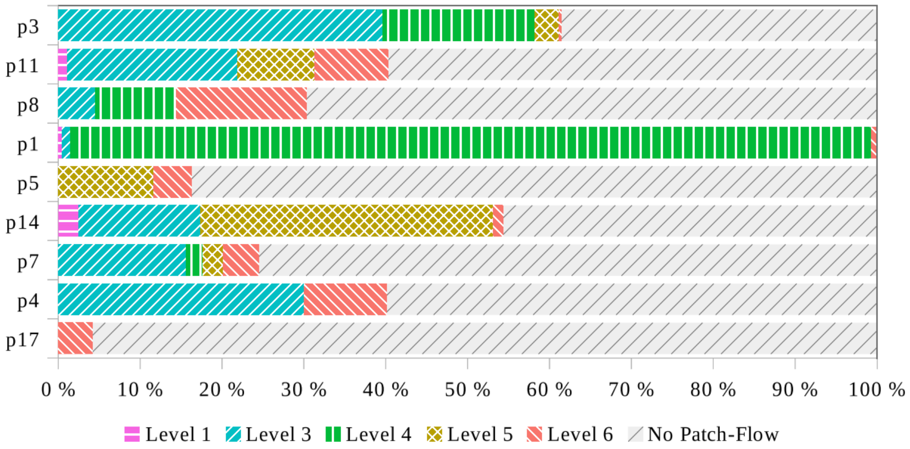


Figure 3.8: Ratio of patches to all code contributions by inner source project



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 3.8 displays only the projects we consider active. The y-axis is ordered by total amount of received code contributions. In contrast to figure 3.7, the bars along the x-axis show the percentage of received patches.

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 org. units (teams) from six level 4 org. 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.

3.5 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.

3.5.1 Evaluation

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

Relevance

In the evaluating case study, we found that about half (47.9%) of all code contributions constitute patch-flow between org. units, almost all (42.2%) being between org. 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 chapter.

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 search for 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 3.4.4) need to carefully re-evaluate who owns this project. In chapter 2, we discussed 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' LCA org. 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 3.2.3: A code contribution from an org. unit to its descendant or ancestor is not patch-flow (because its is not possible that both org. units have a level $n < n_{lca}$ with n_{lca} being the level of their LCA).

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 org. units not participating in IS collaboration. In addition to the patch-flow graph, patch-flow data can show what org. levels are typically crossed by patches, which IS project is developed how actively, and from where each IS project attracts patches.

3.5.2 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 3.4.4). 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 org. unit boundaries within a company. Consequently, where there is a flow of patches (or other contributions), there is IS collaboration.

3.6 Trustworthiness

We evaluate the trustworthiness of our results using the quality criteria credibility, dependability, confirmability and transferability (Guba 1981). These quality criteria for naturalistic research can be applied to evaluate case study inquiries in general (Guba 1981), case studies in the context of software engineering (Cruzes and Dyba 2011), and have been applied to discuss the trustworthiness of case studies that evaluate theories (Stol et al. 2014). 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.

3.6.1 Credibility

Credibility is the degree to which we can establish confidence in the truth of our findings in the context of the inquiry (Guba 1981). For ensuring credibility, we applied two techniques:

We performed intense peer debriefing (Guba 1981): 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 (Hillside-Group 2010) regarding an earlier draft of the research article that is the foundation of this chapter with three researchers.

We performed extensive member checks (Guba 1981): As described in section 3.3.2, we iteratively performed unstructured interviews and a workshop to support our interpretation of the patch-flow data. During interview *I*3 and workshop *W*1, we reflected our findings and interpretations to employees of the studied organizations to discuss our findings and receive feedback which we incorporated into the research article that is the foundation of this chapter.

3.6.2 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 (Guba 1981) 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.

3.6.3 Confirmability

Confirmability is the degree to which we are neutral towards the inquiry and might bias the findings (Guba 1981). As we developed and presented 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 3.6.1 regarding credibility.

3.6.4 Transferability

Transferability is the degree to which findings of our inquiry hold validity in other contexts (Guba 1981). 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.

3.7 Conclusion

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

We evaluated the patch-flow measurement method using case study research. In the case study organization, we found significant patch-flow. This indicates high relevance of the patch-flow phenomenon and justifies the presented method research. We evaluated 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 believe our method is of interest to researchers and practitioners seeking to understand IS collaboration within an organization.

4

Patch-Flow Crawler: A Tool for Measuring Patch-Flow

The previous chapter presented the patch-flow measurement method and found it to be a viable and useful approach to measure the inner source (IS) collaboration within an organization. To use the patch-flow method, researchers need to populate specific data structures (patch-flow graphs and organizational structures as presented in chapter 3) with data from organizations' source code management (SCM) systems and other organizational databases. This can amount to data records on thousands of code contributions authored by thousands of employees from hundreds of different teams. It is difficult (or even impossible) for a researcher to manually collect such data – let alone verify its correctness.

In this chapter, we present the patch-flow crawler – a software tool that implements the patch-flow measurement method. The patch-flow crawler itself is not a research result but an engineering artifact resulting from this thesis. It supports researchers and practitioners in measuring the patch-flow at a software organization. It collects patch-flow data from organizational data sources such as organizations' source code repositories, software forges, and organizational directories. The software tool is implemented in Java.

In detail, this chapter contributes the following:

- The presentation of a the patch-flow crawler (our tool for measuring patch-flow), including …
  - a discussion of its requirements from a researcher's perspective
  - a discussion of its software architecture, design, and implementation
- An evaluation of the fulfillment of the identified requirements

The remainder of this chapter is structured as follows: Section 4.1 defines the requirements towards the patch-flow crawler focusing on the perspective of researchers using the tool. The subsequent sections discuss the tool's software architecture (section 4.2) as well as design and implementation details (section 4.3). Section 4.4 evaluates to which degree the patch-flow crawler satisfies the defined requirements. Section 4.5 closes the chapter with our conclusions.

4.1 Requirements

For collecting and analyzing the requirements towards our patch-flow crawler, we did not follow a specific development process or paradigm. Rather, we synthesized the requirements from the patch-flow method (presented in chapter 3), informal discussions with engineers at three organizations (the three case organizations from chapter 5), and our own industry and IS experiences.

We describe our functional and non-functional requirements using two dimensions:

- Capability: Describing the functionality or non-functional constraint the software tool is required to deliver ("What is the tool supposed to do?")
- Motivation: Describing the benefit achieved by the fulfillment of the requirement ("Why is the tool supposed to do that?")

This format is inspired by the Connexra format for user stories (Lucassen et al. 2016). A user story is a one-sentence requirement documentation with the following format: "As a [role], I want [capability of software], so that [received benefit]". User stories are used as part of eXtreme Programming's planning game (a workshop style approach to effort estimation) (Beck 1999).

In this chapter, we focus on requirements towards the patch-flow crawler by researchers and do not consider stakeholders filling other roles (potentially) interested in the software tool. Because of the small number of requirements, we do not qualify them further by distinguishing requirements that should, need, or have to be fulfilled. We only consider requirements that must be fulfilled.

4.1.1 Overview

Figure 4.1 summarizes our requirements as a hierarchy. The root of the hierarchy is our primary requirement to *extract and persist patch-flow data from organizational data sources*. The leaf nodes are the fine-grained requirements which are further grouped into groups of requirements that are semantically close to one another. In total we identified nine functional requirement (F1-F9) and six non-functional requirements (NF1-NF6).

The functional requirements closely resemble the steps of the patch-flow measurement method presented in chapter 3. However, the patch-flow crawler does not support its users in automatically identifying the organizational unit (org. unit) that is responsible for an IS project (activity 6 of the patch-flow measurement method). We expect researchers need to define their own operational definition of who owns an IS project depending on their research questions or objectives.

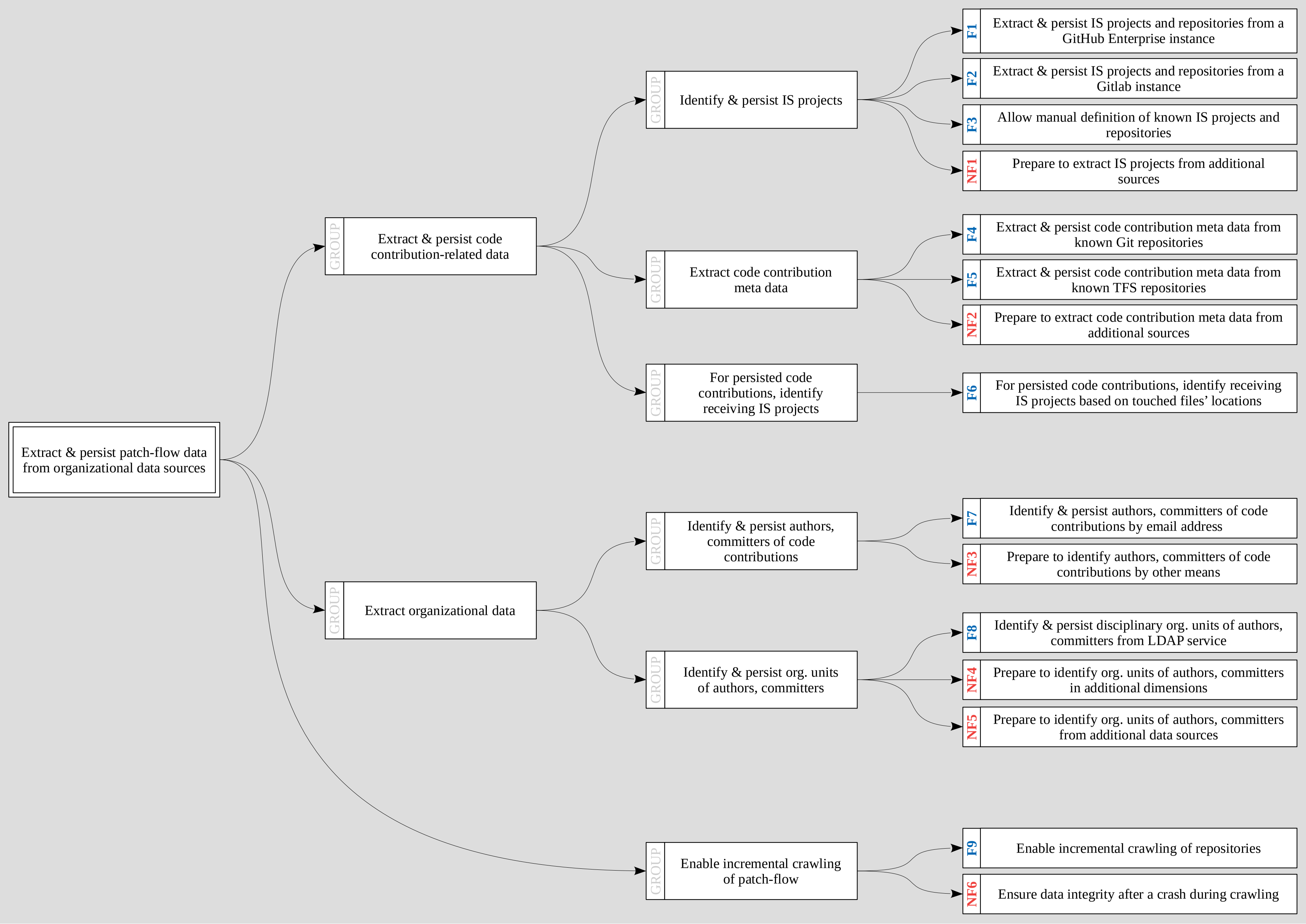
Multiple systems can serve as a data source for extracting patch-flow (or partial patch-flow data). The patch-flow crawler must be capable to extract relevant data from the following systems:

- *GitHub Enterprise & Gitlab* are both software forges (Riehle et al. 2009) that allow individuals to host IS projects. We consider these products, because both have a high prevalence with organizations using IS.

- *Team Foundation Server (TFS)* is a development tool suie developed by Microsoft. It includes an SCM system. We consider this product, because we found it to be used by multiple organizations (developing hardware and software products) in our professional network including two of those studied as part of a multiple-case case study in chapter 5.

- *lightweight directory access protocol (LDAP) services* are used to manage and access directories of an organization's employees and their place within the organizational structure.

Figure 4.1: Hierarchy of requirements towards the patch-flow crawler


Extract & persist patch-flow data from organizational data sources
GROUP
Extract & persist code contribution-related data
GROUP
Extract organizational data
GROUP
Identify & persist IS projects
GROUP
Extract code contribution meta data
GROUP
For persisted code contributions, identify receiving IS projects
GROUP
Identify & persist authors, committers of code contributions
GROUP
Identify & persist org. units of authors, committers
GROUP
Enable incremental crawling of patch-flow
F1
Extract & persist IS projects and repositories from a GitHub Enterprise instance
F2
Extract & persist IS projects and repositories from a Gitlab instance
F3
Allow manual definition of known IS projects and repositories
NF1
Prepare to extract IS projects from additional sources
F4
Extract & persist code contribution meta data from known Git repositories
F5
Extract & persist code contribution meta data from known TFS repositories
NF2
Prepare to extract code contribution meta data from additional sources
F6
For persisted code contributions, identify receiving IS projects based on touched files' locations
F7
Identify & persist authors, committers of code contributions by email address
NF3
Prepare to identify authors, committers of code contributions by other means
F8
Identify & persist disciplinary org. units of authors, committers from LDAP service
NF4
Prepare to identify org. units of authors, committers in additional dimensions
NF5
Prepare to identify org. units of authors, committers from additional data sources
F9
Enable incremental crawling of repositories
NF6
Ensure data integrity after a crash during crawling

We consider such systems as a data source, because they are a de-facto standard in large organizations.

Not all possible data sources for crawling patch-flow, can be known at the initial design time of the patch-flow crawler. Other data sources might become relevant to researchers seeking to measure patch-flow. As a consequence, the crawler has to be designed in such a way, that other data sources can be added without a need to modify existing components of the crawler (NF1, NF2, NF5). A key goal for designing and implementing the patch-flow crawler is to provide a tool that works in execution and domain contexts that might be unknown at design time (NF1, NF2, NF3, NF4, NF5): Researchers need to be enabled to tailor the tool to extract the type of patch-flow data they need to achieve their specific research objectives.

In the following subsections of this section, we will not discuss each fine-grained requirement including its motivation separately due to space constraints. Rather, we will discuss the six lowest level groups in the requirements hierarchy (identify & persist IS projects, extract code contribution meta data, ...).

4.1.2 Identify and Persist Inner Source Projects

To crawl meta data on code contributions and thus patch-flow, one must identify which IS projects are existing and what the coordinates to their source code repositories are. The patch-flow crawler must be able to extract meta data on all IS projects (including the name and coordinates to the repository) stored in a GitHub Enterprise or Gitlab instance (F1, F2) and persist this data. Once a repository or IS project is persisted, we consider it a known IS project or known repository. In case a researcher needs to define manually which projects to include into the scope (e.g. because a particular sampling method is used), the tool has to be capable to allow manual definition of known IS projects and their repositories using a comma separated values (CSV) list (F3).

4.1.3 Extract Code Contribution Meta Data

The patch-flow crawler has to be capable to extract meta data of code contributions (pseudonym of author, pseudonym of committer, commit message, commit date, touched files, type of changes performed) from Git repositories residing in GitHub Enterprise and Gitlab systems (F4). The crawler has to provide the same capabilities for TFS repositories (F5).

These requirements are related to activity 1 of the patch-flow measurement method (extraction of code contributions).

4.1.4 For Persisted Code Contributions, Identify Receiving Inner Source Projects

For each of the identified code contributions, a mapping to the receiving IS project must be performed. The patch-flow crawler has to be capable to map a code contributions to IS projects based on the receiving repository and the files touched by a code contribution (F6). For GitHub Enterprise and Gitlab this is trivial, as we consider repositories and IS projects to be in a 1-to-1 mapping. In TFS instances, multiple IS projects can potentially be hosted in sub directories in the same repository.

This requirement is related to activity 2 of the patch-flow measurement method (mapping of code contributions to IS projects).

4.1.5 Identify and Persist Authors, Committers of Code Contributions

Typically, each code contribution carries some information on its author (and committer). For example, a commit in Git typically has an author pseudonym field with contents in the following format: "Jonathan Doe <email@address.org>". The patch-flow crawler has to be capable to parse such pseudonym strings and identify authors using their email address (F7). For some studies such an identification might either not be feasible or not sufficient. It has to be possible to adapt the patch-flow crawler to use other means to identify authors without modifying existing components of the tool (NF3).

These requirements are related to activity 4 of the patch-flow measurement method (identification of author).

4.1.6 Identify and Persist Org. Units of Authors, Committers

For the identified authors and committers, the patch-flow crawler must be capable to identify their org. unit in the disciplinary organization using an LDAP service (F8). Depending on the research questions and objectives, researchers studying the patch-flow might be interested in more than the organization's disciplinary structure. Researchers could be interested in ...

- … in the regional structure of an organization (for example to study IS's effects on globally distributed software development).
- … the legal entity structure of an organization (for example to study IS's implications on controlling and taxation).
- … in the (traditional) project organization (for example to study IS's effects on project management).
- … other formal and informal dimensions of organizational structure.

As a consequence, not only must the patch-flow crawler's design be prepared to extract organizational data from additional data sources (NF5) but also for additional dimensions of the organizational structure not known at initial design time (NF4). These requirements are related to activities 3 and 5 of the patch-flow measurement method (extraction of organizational data, mapping of authors to org. units).

4.1.7 Enable Incremental Crawling

Developers can potentially contribute hundred thousands of code contributions to the IS projects of an organization's IS program. Extracting patch-flow data for such big programs could take a long time (multiple hours or days) and it might not be possible to crawl it within one run of the patch-flow crawler.

For example, it happened to us that we – as external researchers – were not given our own machines with access to the company network but rather relied on shared machines at the organizations' campuses where we could get access for blocks of a few hours or days. Due to such situations, the patch-flow crawler has to be able to perform the crawling incrementally: That is, if the crawler is interrupted, it needs to be capable to resume from and still consider the data that had been crawled already (F9). If the patch-flow crawler is interrupted or crashes for any reason the integrity of already crawled and persisted data must not be affected (NF6).

4.2 Software Architecture

To implement the requirements, we designed and implemented the patch-flow crawler. This section describes its software architecture.

The IEEE standard 1471-2000 defined software architecture as "the fundamental organization of a [software] system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution" (as quoted by Reussner and Hasselbring 2006). Describing, documenting, and discussing software architecture is a non-trivial undertaking. Prior work has presented multiple frameworks for presenting software architectures (Kruchten 1995; Reussner and Hasselbring 2006). Reussner and Hasselbring (2006) propose to describe software architecture using three distinct views (or how they call it "standpoints"):

- The *static view* describes the decomposition of a system in its elements and their dependencies.
- The *dynamic view* describes the system's behavior at run time, including its control flow, and changes and interactions between the system's components.
- The *deployment view* describes how elements of the static view are mapped to specific infrastructure and hardware units.

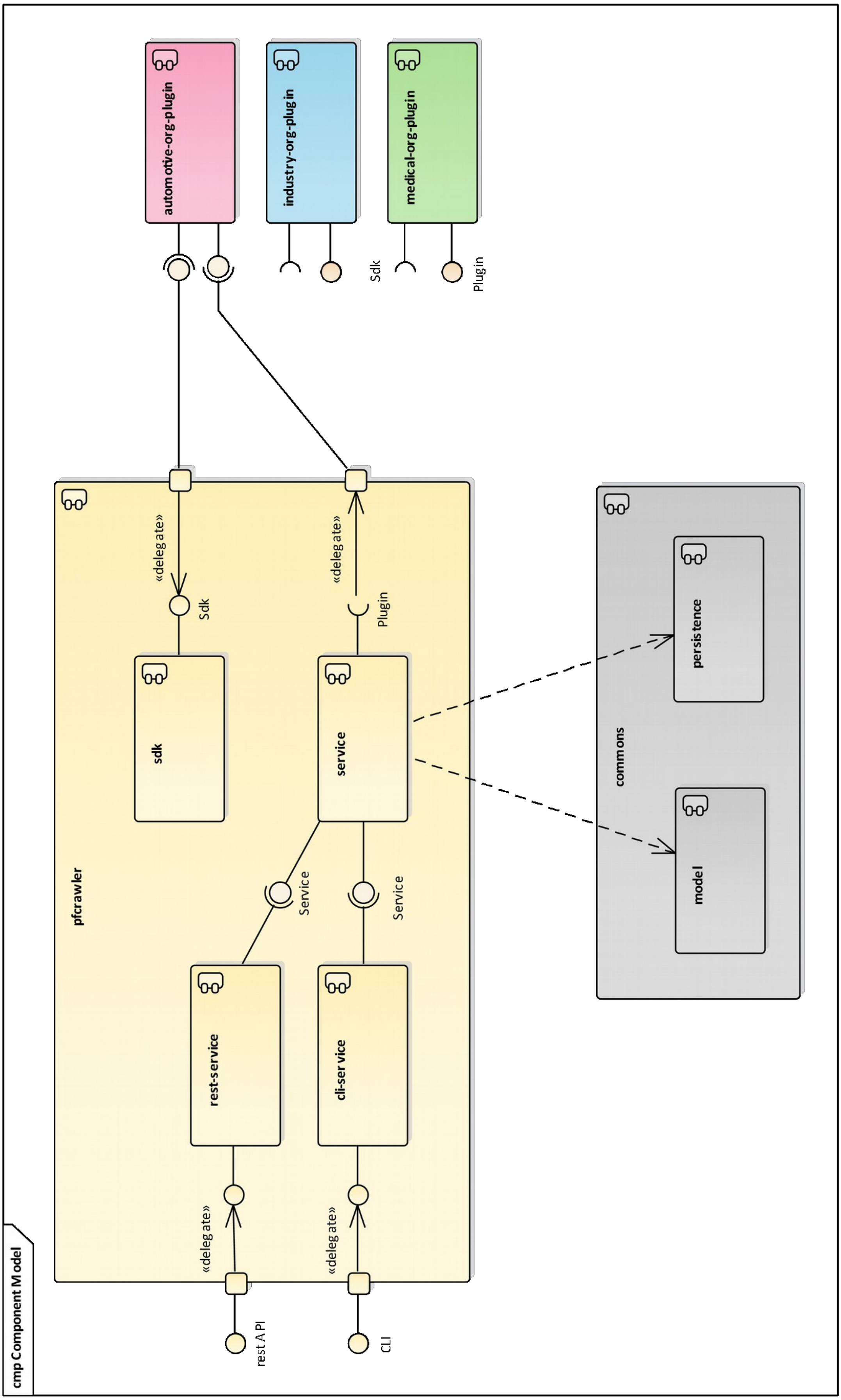
The next subsections discuss the patch-flow crawler's architecture from these three views.

4.2.1 Static View

Figure 4.2 summarizes the high level logical structure of the patch-flow crawler as UML 2 component diagram. The colors are used to highlight the different high-level components and carry no further semantics. The visualized components encapsulate the following responsibilities:

- The component *pfcrawler* encapsulates the functionality specific to the patch-flow crawler. It contains multiple sub-components:
  - The component *service* encapsulates all functionality for initiating a run of the patch-flow crawler. It delegates to a **-plugin* component to communicate with patch-flow data sources (like organizational databases or SCM systems) and makes use of the commons component's functionalities to persist the patch-flow data.
  - The components *rest-service* and *cli-service* provide functionality to use the patch-flow crawler via a REST API (Fielding 2000) or a command line interface (CLI).

Figure 4.2: Logical components of the patch-flow crawler as UML 2.0 component diagram



    - The component *sdk* provides a software development kit that other researchers can use to implement their own plugins defining how to crawl patch-flow in a specific environment (for example within a specific organization).

- The **-plugin* components each contain the functionality for one such plugin. Organizations differ from one another. While one company might use Git to manage their source code repositories, another might use a tool not known at initial design time of the patch-flow crawler (see non-functional requirements NF1, NF2, NF5). When crawling patch-flow at a specific organization, researchers or developers can implement a plugin component enabling them to collect the necessary patch-flow data from the specific organization's data sources. The service component orchestrates the lower level functionalities of a **-plugin* component hidden behind the Plugin interface. This interface and its contract will be discussed in section 4.3.3

- The *commons* component contains functionality that is not specific to the patch-flow crawler but might also be used by other software programs relying on patch-flow data. It contains two sub-components:

    - The component *model* contains a domain model allowing to represent an organization and its patch-flow.

    - The component *persistence* provides functionality to persist patch-flow and organizational data in a relational database and retrieve it.

The patch-flow crawler cannot run without a plugin because the *pfcrawler* component consumes the *Plugin* interface. Thus, one could argue that the patch-flow crawler in itself is not a tool but a framework for patch-flow crawling. Riehle (2000) defines a framework as follows:

> A framework "is a reusable design together with an implementation [...]. The design represents a model of an application domain or a pertinent aspect thereof, and the implementation defines how this model can be executed, at least partially. [...] It leaves enough room for customization to solve a particular problem in the application domain."

We still consider the patch-flow crawler a complete tool because it comes with a set of default plugins researchers can use. However, developers extending the patch-flow crawler might very well consider it a framework.

4.2.2 Dynamic View

Every execution of the patch-flow crawler is done as part of a crawl run. A crawl run is the sequence of all activities performed by the patch-flow crawler to extract patch-flow data for all considered source code repositories. A crawl run ends if patch-flow extraction for all considered source code repositories has either succeeded or failed or the crawl run was interrupted. Each crawl run has a specific state. It can be either ...

- ... initialized (just started or currently running)
- ... crawled (successfully finished)
- ... failed

Incremental Crawling

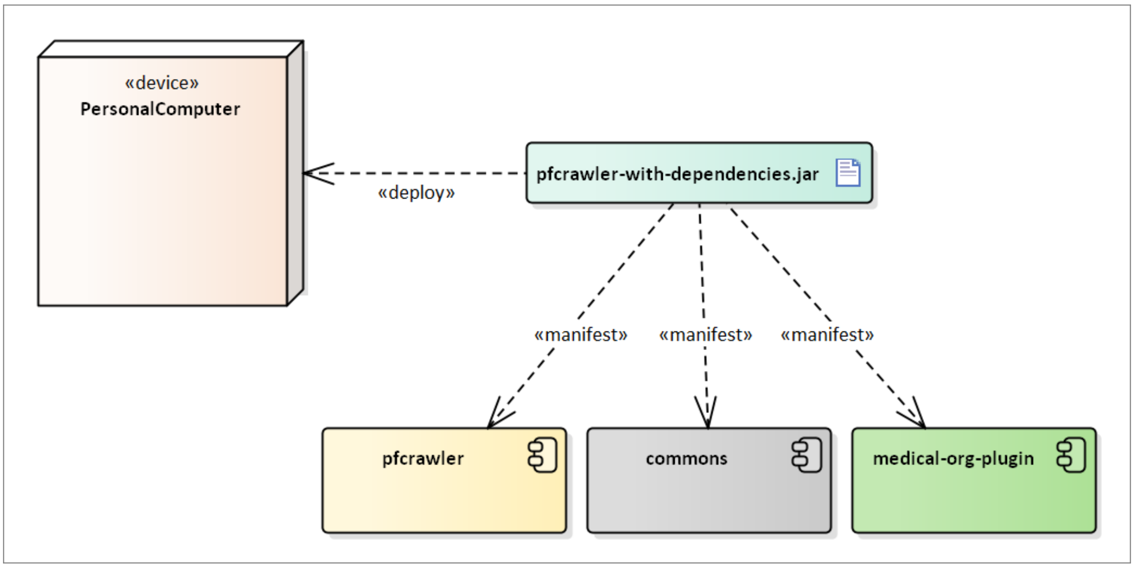
A code contribution persisted as a commit to a source code repository is considered to be immutable. Its contents or meta data do not change. As a consequence, we can enable incremental crawling by extracting for each source code repository only the patch-flow data for a defined time interval. Because this approach is similar to the one presented in Capraro (2013), we will only briefly discuss it here:

- Initial crawling. When a repository is crawled for the first time, code contributions contributed in the time interval $[null, t_1)$ will be extracted (with t_1 being the time the first crawl run was initiated), while *null* is interpreted as beginning of time. For each repository, the last successful crawl run will be persisted (including its t_1).
- Incremental crawling. In the subsequent crawl run, code contributions for the time interval $[t_1, t_2)$ will be extracted (with t_2 being the starting time of the second crawl run).

If a crawl job n started at t_n for a repository fails, this repository's last successful crawl run will not be changed. In the next crawl run $n+1$, patch-flow data for the time interval $[t_{n-1}, t_{n+1})$ will be extracted from the repository (if the repository was crawled successfully in crawl run $n - 1$).

If no crawl run was successful yet (be it the first or multiple failed) for a repository, the a following crawl run k will attempt to crawl patch-flow data for the time interval $[null, t_k)$ for that repository.

Figure 4.3: Deployment of the patch-flow crawler as command line tool as UML 2.0 deployment diagram



4.2.3 Deployment View

In the default deployment scenario, the crawler is deployed as a standalone application with a CLI to a researcher's machine. However, we prepared the crawler for alternative deployment scenarios.

Deployment as Command Line Tool

Figure 4.3 summarizes a deployment of the patch-flow crawler as a CLI application with a UML 2.0 deployment diagram. The *pfcrawler* and *commons* components as well as one patch-flow crawler plugin are packaged into a Java **.jar* file. This file is deployed to a personal computer from where a researcher can use it to extract patch-flow data.

Deployment as REST Service

We envision that the patch-flow crawler might be used not only by researchers but also as part of a larger set of software components for extracting IS data, calculating metrics, and visualizing IS collaboration. In such a scenario, it might not be sufficient to manually trigger patch-flow crawl runs using a CLI. The patch-flow crawler can be configured to be controlled via a simple

REST API that can initiate a new crawl run (*POST /crawlruns*) or retrieve information on the status of a crawl run (*GET /crawlruns/:id*).

Deployment on Multiple Devices

With patch-flow crawling potentially taking a long time, we envision that users of the patch-flow crawler might eventually want to distribute patch-flow crawling across multiple devices. While the patch-flow crawler is currently not capable to distribute its work on to multiple machines, we prepared it to allow for such an extension in the future: In section 4.3.2, we will describe how we encapsulated specific tasks into the CrawlJob class. In future implementations, such crawl jobs could be distributed to multiple devices.

4.3 Design & Implementation

The last section delivered a coarse-grained top down view on the patch-flow crawler's software architecture. This section will discuss selected fine-grained design and implementation details.

4.3.1 Domain Model

For representing patch-flow data, we developed an object-oriented domain model. We need to model the relationship of code contributions to elements of the organization (for example authors of code contributions, their org. units, or IS projects receiving code contributions).

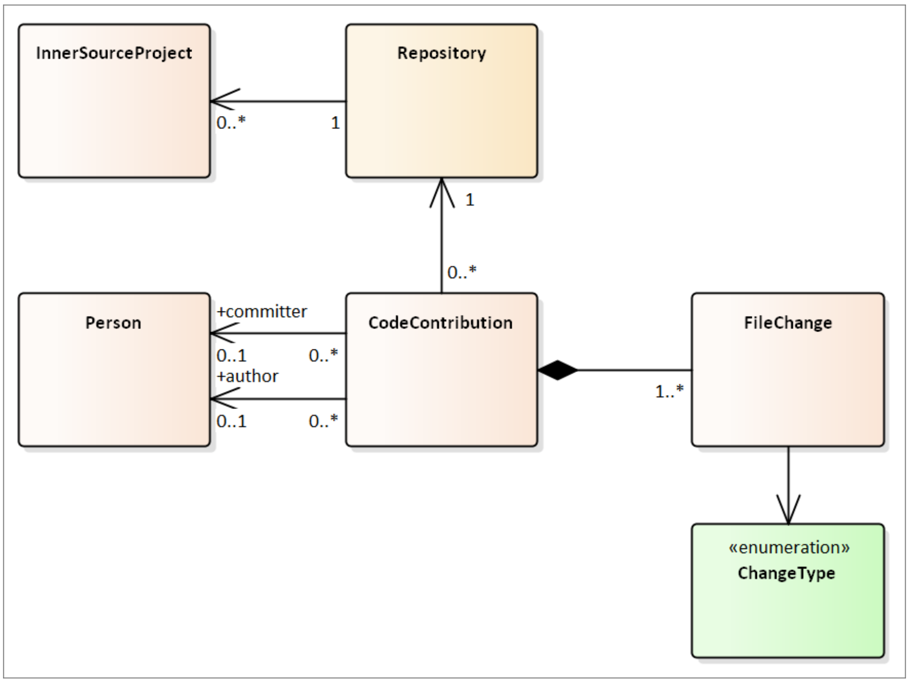
Code Contributions and Patch-Flow

Figure 4.4 shows the part of the domain model representing a code contribution and classes it is associated with as a UML 2 class diagram.

A code contribution (*CodeContribution*) is composed of multiple file changes (*FileChange*) that each represent one file touched by the code contribution. In the context of the patch-flow crawler, a file change cannot exist without a code contribution. A file change has a change type (*ChangeType*) expressing whether the file was modified, created, or deleted as part of the code contribution.

A code contribution is posted to a repository (*Repository*). One repository can contain zero, one, or multiple IS projects (*InnerSourceProject*). One IS project cannot have more than one

Figure 4.4: Code contributions and associated classes as UML 2.0 class diagram



repository. While software forges like GitHub or Gitlab encourage a one-to-one mapping between IS projects and repositories, other setups can lead to multiple IS projects being hosted in one repository.

Each code contribution has zero or one author and zero or one committer (both objects of the class *Person*). One author and committer is the default case. However, sometimes it is not possible to identify an author or committer object (for example when sufficient organizational data is not available or when repositories have been migrated from one SCM to another).

The concepts and terminologies of our domain model (such as persons, repositories, ...) are different from the concepts and terminologies in the context of the systems we use as data sources. Appendix B provides a mapping of terms in our domain model to terms in GitHub, Gitlab, and TFS.

Accountability and Typed Relationship Patterns

In addition to modeling code contributions and the concepts they are directly associated with, we must model the elements and structure of the organizations whose patch-flow is measured. To model organizational structures, we rely on the *typed relationship* pattern (Fowler 1997b).

Figure 4.5: Typed relationship pattern as UML 2.0 class diagram

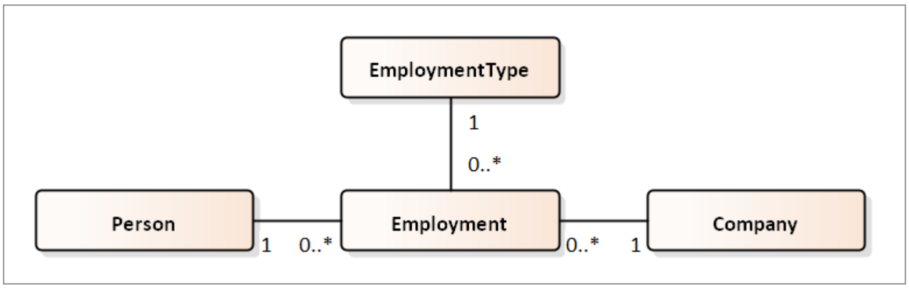
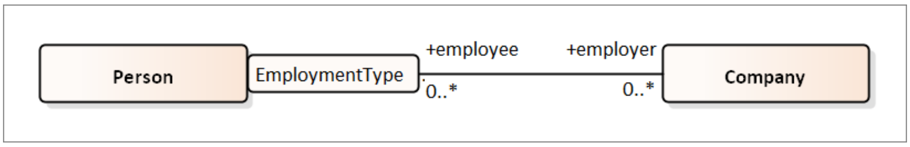


Figure 4.6: Abbreviated notation for typed relationship pattern as UML 2.0 class diagram



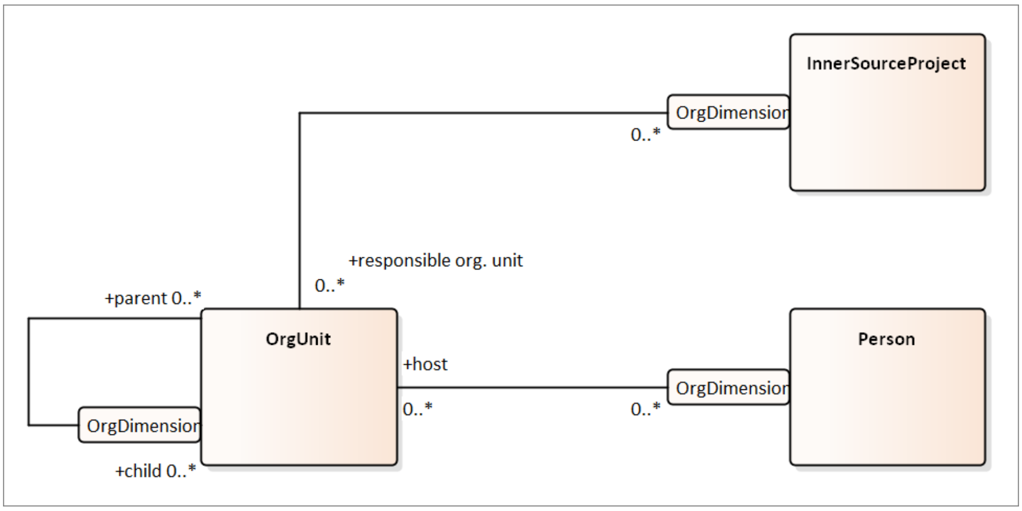
A simple approach to modeling organizations is to model them as trees composed of disciplinary org. units, for example using the composite design pattern proposed by Gamma et al. (1994). However, this approach is not sufficient because we need to be capable to model multiple dimensions of the organizational structure like the disciplinary, regional, and legal-entity dimension (see non-functional requirement NF4).

This can be achieved using the *typed relationship* design pattern introduced by Fowler (1997b). Figure 4.5 gives Fowler's initial example of the typed relationship design pattern as UML 2.0 class diagram. In the example, a person (*Person*) is employed (*Employment*) at a company (*Company*). To further qualify the employment relationship, the type of employment (*EmploymentType*) is associated with the employment. In this thesis, we use UML 2.0 qualified associations to express typed relationships. Figure 4.6 shows Fowler's example using this notation. The use of typed relationship for modeling organizational or reporting structures results in the accountability design pattern (Fowler 1997a).

Organizational Structure

The domain model of the patch-flow crawler makes use of the *typed relationship* design pattern by modeling the relationships between org. units and persons, IS projects, and other org. units as typed relationships. Each such relationship holds validity in one defined organizational dimension (*OrgDimension*) it is associated with. An organizational dimension is the dimension along which an organization is decomposed. For example one can decompose an organization

Figure 4.7: Organizational units and associated classes as UML 2.0 class diagram



according to its disciplinary or functional structure, its legal or regional entities. Figure 4.7 summarizes the organizational model as a UML 2.0 class diagram.

4.3.2 Crawl Engine

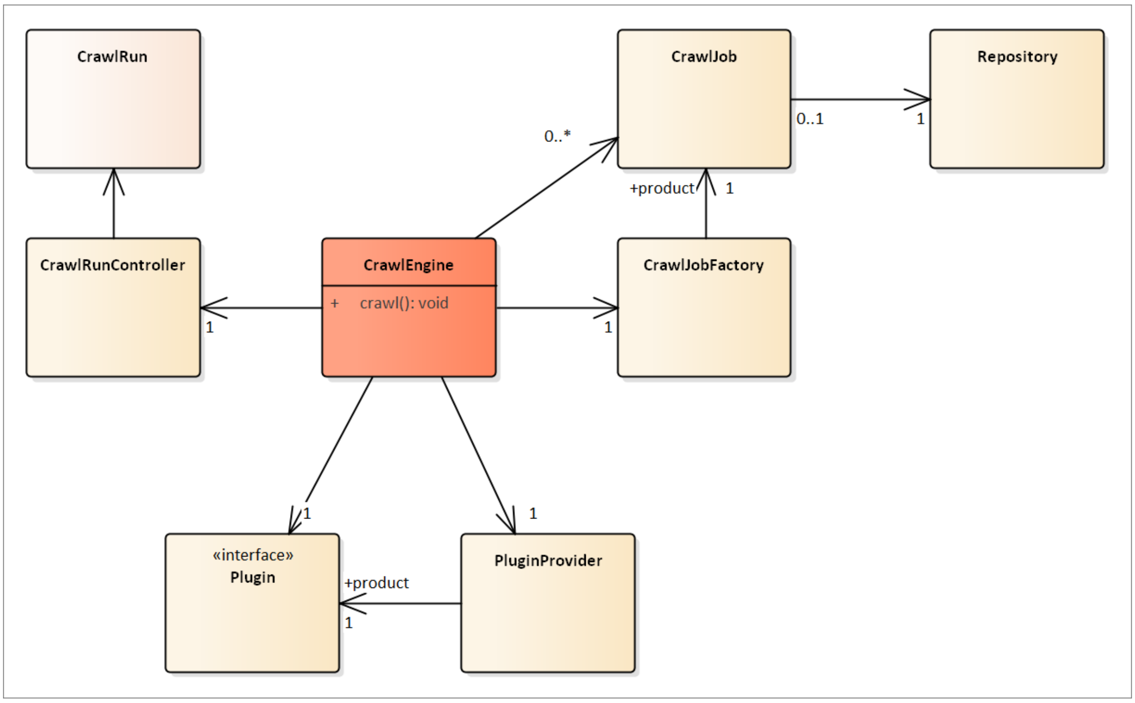
One central class of the patch-flow crawler's *service* component is the crawl engine (*CrawlEngine*). It orchestrates all tasks necessary to perform the patch-flow crawling. Figure 4.8 summarizes this class (highlighted in red color) and other selected classes it is associated with:

- *CrawlJobFactory and CrawlJob.* For each repository (*Repository*) to crawl, the crawl engine uses the crawl job factory (*CrawlJobFactory*) to instantiate one crawl job (*CrawlJob*). The crawl engine schedules and monitors the crawl jobs. Crawl jobs perform the resource intense work of communicating with the data sources and transforming the extracted data into the required format. Encapsulating this functionality into its own class allows future researchers and developers to implement alternative deployment scenarios (see section 4.2.3)

- *CrawlRunController.* The crawl engine delegates to the crawl run controller (*CrawlRunController*) all activities necessary to manage the state of the current crawl run.

- *PluginProvider and Plugin.* A plugin provider (*PluginProvider*) provides to the crawl engine an instance of a class implementing the *Plugin* interface. This allows researchers and developers to customize a crawl run.

Figure 4.8: Crawl engine and associated classes as UML 2.0 class diagram



4.3.3 Plugin Interface

Researchers using the patch-flow crawler, can customize which tasks are executed during a crawl run by implementing the patch-flow crawler's Plugin interface. The following listing shows the *Plugin* interface as Java source code:

```
public interface Plugin {
        List<CodeContributionProcessor> getCodeContributionProcessors();
        ScmAdapter getScmAdapter(Repository _repository);
        List<PreStep> getPreSteps();
        List<PostStep> getPostSteps();
}
```

By implementing the methods of the Plugin interface, researchers and developers decide which tasks should be performed as part of a crawl run.

Pre- and Post-Steps

The plugin provides to the crawl engine a (sorted) list of pre-steps and post-steps. Pre-steps (implementing the *PreStep* interface) encapsulate tasks that need to be performed before the crawl jobs can commence. For example, one could use a pre-step to identify whether new IS projects or repositories were added to a software forge that need to be considered when crawling. After all crawl jobs finished, the crawl engine executes a sequence of post-steps (implementing the *PostStep* interface). The sequence of steps is determined by their position in the the list (implementing Java's *List*<> interface). The post-steps encapsulate tasks that need to be performed before the crawl run ends, for example cleanup tasks or anonymization of extracted data.

Source Code Management Adapters

To extract code contributions, the crawl engine delegates to SCM adapters (implementing the *ScmAdapter* interface). Using the *getScmAdapter()* method of the plugin, the crawl engine can get the needed SCM adapter for each repository it seeks to instantiate a crawl job for. The following source code listing shows the *ScmAdapter* interface:

```
public interface ScmAdapter {
        // Dropped comments from this listing
        public Iterator<CodeContribution> fetch(
                Repository _repository,
                Date _start,
                Date _end
        );
}
```

The *fetch()* method of each SCM adapter guarantees to return an iterator (Gamma et al. 1994) of code contributions sorted by contribution date. The sorting ensures that if a crawl job is interrupted the last persisted code contribution and all code contributions before that can still be used to construct a complete time series of code contributions per repository up until the contribution time of the last persisted code contribution. Using Java's iterator interface instead of Java's list implementation, allows developers implementing an SCM adapter to optimize the resource consumption by using lazy loading techniques.

Code Contribution Processors

Each code contribution extracted using SCM adapters is processed by a chain of code contribution processors (implementing the *CodeContributionProcessor* interface) objects. Such processors can for example be used to extract additional information about the code contribution or its author from an organizational database. Similarly to the

The patch-flow crawler's *sdk* component provides a set of pre-steps, post-steps, code contribution processors, and SCM adapters. Users can also provide their own implementation by implementing the respective interfaces (*PreStep*, *PostStep*, *CodeContributionProcessor*, *ScmAdapter*). The source code of all interface definitions is listed in appendix B.

Example Plugin

Figure 4.9 illustrates a crawl run using an example plugin (the plugin for the automotive organization we will study in chapter 5) as a UML 2.0 communication diagram. The diagram omits objects that are not necessary for discussing the sequence of events in a crawl run.

Once a user calls the crawl engine's *crawl()* method (message 1 in the communication diagram), the crawl engine uses a provided plugin (not pictured) to instantiate the pre-steps, post-steps, and patch-processors this plugin defines. First a pre-step is executed by calling the execute() method of an object instantiating the *AddInnerSourceProjectsPreStep* (message 1.1). The class encapsulates functionality for identifying IS projects (and their repositories) that were newly created since the last crawl run and persisting them as known IS projects (and repositories).

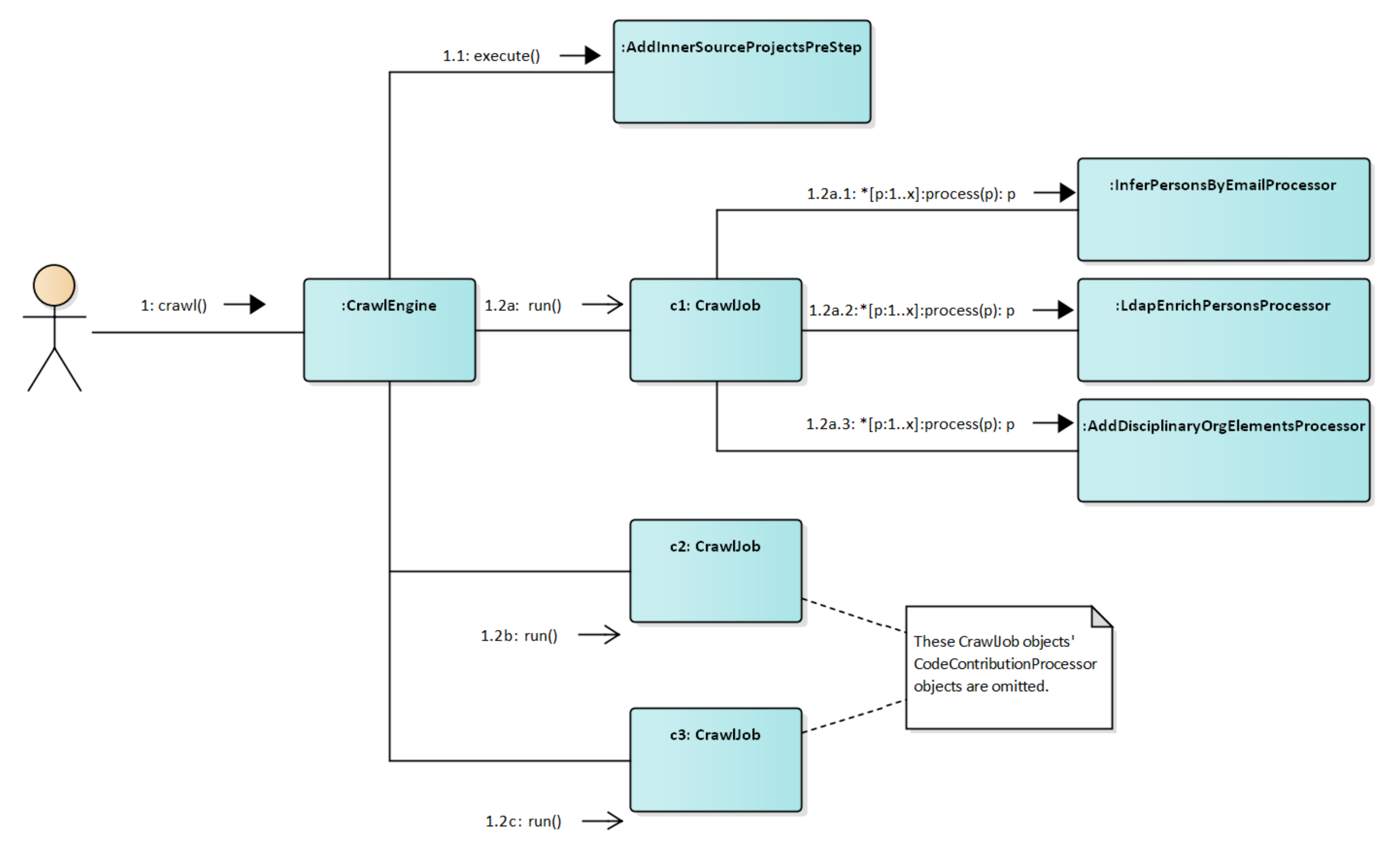
For each known repository, the crawl engine creates one instance of the class *CrawlJob*. In this example, three crawl jobs are created and run independent and in parallel (messages 1.2a, 1.2b, 1.2c).

The crawl job extracts data about code contributions from its repository and creates multiple instances of the *CodeContribution* class. Each such instance *p* (assuming *x* instances) is processed with a sequence of (in this example three) code contribution processors:

- The *InferPersonByEmailProcessor* identifies who contributed a code contribution by extracting the email address of the author and committer from the code contribution meta data (message 1.2a.1).

- The *LdapEnrichPersonProcessor* connects to an LDAP service to load additional data on the code contribution's author and committer for example first name and last name (message 1.2a.2).

Figure 4.9: Object communication during a crawl run as UML 2.0 communication diagram


1: crawl()
:CrawlEngine
1.1: execute()
:AddInnerSourceProjectsPreStep
1.2a: run()
c1: CrawlJob
1.2a.1: *[p:1..x]:process(p): p
:InferPersonsByEmailProcessor
1.2a.2:*[p:1..x]:process(p): p
:LdapEnrichPersonsProcessor
1.2a.3: *[p:1..x]:process(p): p
:AddDisciplinaryOrgElementsProcessor
c2: CrawlJob
1.2b: run()
c3: CrawlJob
1.2c: run()
These CrawlJob objects' CodeContributionProcessor objects are omitted.

- The *AddDisciplinaryOrgElementsProcessor* again connects to an LDAP service to identify the code contribution author's org. unit and this org. unit's parents in the disciplinary organizational tree (message 1.2a.3).

4.4 Evaluation

This section discusses how we evaluated the fulfillment of the identified requirements.

Table 4.1 gives a summary. It lists all requirements and describes how they were we evaluated their fulfillment. Ten requirements (F1, F2, F3, F4, F5, F6, F7, F8, F9, NF6) were evaluated by performing software tests. Typically, this tests were performed as part of a master thesis project where a student contributed to the development of the patch-flow crawler. Five requirements (NF1, NF2, NF3, NF4, NF5) were not evaluated by a software test. Rather, we evaluated them analytically by studying the properties of the software design.

4.4.1 GitHub Enterprise-Specific Requirements

As part of the master thesis project by Plantera (2018), we evaluated the patch-flow crawler's capabilities to crawl patch-flow from GitHub Enterprise and implemented GitHub-specific SCM adapters, code contribution processors, and pre-steps. For evaluation, we extracted patch-flow data using the API of GitHub.com and a mocked LDAP service. We manually inspected and checked the extracted data. The API of GitHub.com is largely identical to the API of GitHub Enterprise. We found that the patch-flow crawler is capable to extract and persist IS projects from a GitHub Enterprise instance (F1), to extract code contribution meta data from Git repositories (F4), to identify authors based on email addresses (F7), and to identify and persist disciplinary org. units of persons using an LDAP service (F8).

4.4.2 Gitlab-Specific Requirements

As part of the master thesis project by Metzig (2019), we evaluated the patch-flow crawler's capabilities to crawl patch-flow from a Gitlab forge and implemented Gitlab-specific SCM adapters, code contribution processors, and pre-steps. For evaluation, we extracted code contribution data from a Gitlab instance hosted and provided by the RRZE – our university's central provider of IT infrastructure. We found the crawler to be capable to extract and persist

Table 4.1: Overview of requirement evaluation status

Status	Requirement	
Identify & persist IS projects		
✔ tested	F1	Extract & persist IS projects and repositories from a GitHub Enterprise instance
✔ tested	F2	Extract & persist IS projects and repositories from a Gitlab instance
✔ tested	F3	Allow manual definition of known IS projects and repositories
✔ eval. analytically	NF1	Prepare to extract IS projects from additional sources
Extract code contribution meta data		
✔ tested	F4	Extract & persist code contribution meta data from known Git repositories
✔ tested	F5	Extract & persist code contribution meta data from known TFS repositories
✔ eval. analytically	NF2	Prepare to extract code contribution meta data from additional sources
For persisted code contributions, identify receiving IS projects		
✔ tested	F6	For persisted code contributions, identify receiving IS projects based on touched files' locations
Identify & persist authors, committers of code contributions		
✔ tested	F7	Identify & persist authors, committers of code contributions by email address
✔ eval. analytically	NF3	Prepare to identify authors, committers of code contributions by other means
Identify & persist org. units of authors, committers		
✔ tested	F8	Identify & persist disciplinary org. units of authors, committers from LDAP service
✔ eval. analytically	NF4	Prepare to identify org. units of authors, committers in additional dimensions
✔ eval. analytically	NF5	Prepare to identify org. units of authors, committers from additional data sources
Enable incremental crawling of patch-flow		
✔ tested	F9	Enable incremental crawling of repositories
✔ tested	NF6	Ensure data integrity after a crash during crawling

IS projects from a Gitlab instance (F2) and to extract and persist code contribution meta data from Git repositories (F4).

4.4.3 TFS-Specific Requirements

As part of the master thesis project by Hasler (2017), we evaluated the patch-flow crawler's capabilities to crawl patch-flow from a TFS instance and implemented a TFS specific SCM adapter. For evaluation, we used the patch-flow crawler with this SCM adapter, to extract code contribution data from a TFS repository hosted as a software as a service. We found the crawler to be capable to extract and persist code contribution meta data from TFS repositories (F5). We were able to manually provide the TFS repositories to the crawler (F3).

4.4.4 Remaining Requirements

All three cited master thesis projects made use of the crawler's capability to crawl incrementally (F9) and tested the crawler's behavior when interrupted. We found that already crawled data's integrity remains intact even if the crawler is interrupted unexpectedly (NF6).

For five non-functional requirements (NF1, NF2, NF3, NF4, NF5), we could not perform an evaluation by testing the patch-flow crawler or its output. Rather, these non-functional requirements are fulfilled as a consequence of specific design decisions: Using the typed property pattern to model relationships between org. units, allows our domain model to express multiple organizational dimensions defined at run time (NF4). The plugin interface allows developers to customize the crawler to extract IS projects and coordinates to repositories from arbitrary sources (NF1), to identify code contributions or their authors by alternative means (NF2, NF3), or to use arbitrary systems as data sources to identify the org. units of code contribution authors (NF5).

4.5 Conclusion

In this chapter, we presented the patch-flow crawler – a Java tool for crawling patch-flow – and discussed its requirements, software architecture, design, and implementation. The tool allows researchers (and others) to automate the patch-flow measurement method presented in chapter 3. The tool can be used to crawl patch-flow from multiple software forges and SCM systems (for example GitHub Enterprise, Gitlab, TFS) as well as organizational data sources (for example LDAP services). Using a plugin interface, the crawler can be customized to extract patch-flow data from arbitrary data sources.

5

Case Study: Patch-Flow at Three Large Organizations

The majority of scientific literature presented qualitative results such as case study reports (for example Dinkelacker et al. (2002); Gurbani et al. (2006); Stol et al. (2014); Riehle et al. (2016)) and taxonomies or qualitative models regarding inner source (IS) (for example Stol et al. (2011, 2014); Gaughan et al. (2007)). There is no study yet quantifying the maginitude of IS collaboration or exploring the relationship between exercised IS practices and resulting IS collaboration.

Learning about the magnitude of IS collaboration is relevant for IS practitioners and researchers because it tells how much IS collaboration is to be expected in organizations. The magnitude indicates the relevance of research on IS proxied by IS's possible impact on industry organizations. Understanding how implemented IS practices affect IS collaboration is of interest to to primarily practitioners. It indicates whether the cost of adopting specific IS practices is justified because it leads to a different magnitude or properties of IS collaboration.

To this end, this chapter answers the following research questions:

- RQ4: What is the magnitude of IS collaboration in organizations?
- RQ5: How do IS practices affect IS collaboration?

To answer these research questions, we performed a multiple-case case study with three software developing organizations running five IS programs. We performed qualitative and quantitative analyses:

In the *qualitative* part, we employed multiple data collection methods (resulting in 14 interviews, 17 direct observation notes, 15 documents, 10 artifacts) and analyzed the resulting data using thematic analysis (Braun and Clarke 2006) to learn which IS practices are implemented in the organizations.

In the *quantitative* part, we measured and analyzed the patch-flow in the organizations. Patch-flow is the flow of code contributions across organizational boundaries such as project or organizational unit (org. unit) boundaries within a company. Patch-flow uses patches (external code contributions) as a proxy to model collaboration. Analyzing patch-flow tells us who in an organization contributed how many code contributions to whose IS projects. We identified correlations between the observed IS practices and patch-flow and theorize about relationships between them.

In detail, this chapter contributes the following:

- A multiple-case case study with three software developing organizations, including ...
    - a discussion of how IS is implemented in three case organizations
    - an in-depth analysis of the case organizations' patch-flow
- A theory (consisting of four hypotheses) on how IS practices affect IS collaboration

The remainder of this chapter is structured as follows: Section 5.1 gives an overview of prior work regarding IS and the effect of IS practices on IS collaboration. Section 5.2 introduces our research methods by discussing our case study design and execution as well as the approach for measuring patch-flow. The subsequent sections report on our case study results by giving case descriptions (section 5.3) and presenting our cross-unit of analysis synthesis (section 5.4). Section 5.5 discusses and interprets the findings of our case study and presents four hypotheses forming a theory on how IS practices affect IS collaboration. Section 5.6 discusses the trustworthiness of our our study and section 5.7 presents our conclusions.

5.1 Related Work

In this section, we discuss prior work related to our study. First, we scope our case study from other case studies performed in an IS context. Second, we discuss related work on quantifying the magnitude of IS collaboration (RQ4) and, third, on the influence of IS practices on IS collaboration (RQ5).

5.1.1 Prior Case Studies

The majority of prior work on IS to date is relying mostly on qualitative data. A variety of case studies reported on IS efforts and programs organizations including Bosch (Cooper and Stol 2018), DTE Energy (Smith and Garber-Brown 2007), Ericsson (Torkar et al. 2011), Hewlett-Packard (Dinkelacker et al. 2002; Melian and Mähring 2008), IBM (Vitharana et al. 2010), Kitware (Martin and Hoffman 2007), Lucent (Gurbani et al. 2006, 2010), Nokia (Lindman et al. 2008), (Cooper and Stol 2018), Philips (van der Linden et al. 2009), Rolls-Royce (Stol et al. 2014), and SAP (Riehle et al. 2009). None of these studies quantified IS collaboration or explored the relationship between the observed IS practices and the resulting IS collaboration.

However, some prior work utilized quantitative data as well. Both, Dinkelacker et al. (2002) and Riehle et al. (2009) used the number of IS projects to quantify the size of an IS program. Contrary to our work, they did not consider whether the IS projects were receiving any outside contributions. Gurbani et al. (2006) measured counted all code contributions to an IS project over time to illustrate the project's evolution. Contrary to our work, they did not distinguish between outside contributions (patches) and regular code contributions.

5.1.2 Magnitude of IS Collaboration

Both researchers and practitioners have used simple counting metrics to quantify how active their IS programs are and how much collaboration are taking place. We discussed their approaches in detail section 3.1 (discussing prior work related to the patch-flow method). The approach of our study differs in that we consider the organizational structure of the studied organizations and the flow of code contributions between its org. units.

5.1.3 Influence of IS Practices

Prior work did not explicitly study the scope, extent, or magnitude to which IS practices influence IS collaboration. However, implicitly many studies present approaches, factors, and insights with the goal to benefit or increase IS collaboration.

Stol et al. (2014) used prior literature to identify nine key factors that support IS adoption. Stol and Fitzgerald (2015) presents a tutorial on adopting IS based on these key factors. Our work is similar to their work in that we also build a theory, seeking to understand what leads to or enables more IS collaboration. However, our work differs in that we use rich quantitative data (patch-flow data) to build our theory. Our findings are more fine-grained. We theorize on the influence of specific practices or sets of practices and consider the magnitude of collaboration (proxied by patch-flow) and the organizational distance of participants in collaboration.

Carroll et al. (2018) examine the impact of adopting IS practices on organizations. They used qualitative methods to identify tensions arising from using IS practices and propose strategies to mitigate these tensions. In contrast, we theorize how specific sets of IS practices affect IS collaboration.

5.2 Research Approach

We performed case study research. We identified which IS practices the case organizations implemented, measured and analyzed the patch-flow (as a proxy for IS collaboration), identified correlations between employeed IS practices and resulting IS collaboration, and theorized about their causal relationships.

Case study research is the appropriate research method because we studied a "contemporary phenomenon [...] within its real world context" (Yin 2013). Different classes of case studies exist (Yin 2013; Runeson et al. 2012). Our case study is ...

- a *multiple-case* case study (Yin 2013), because we studied three organizations with an isolated context.

- an *embedded* case study (Yin 2013), because we consider the organizations' IS programs to be units of analysis (two organizations run more than one IS program).

- an *exploratory* case study (Runeson et al. 2012), because we undertake theory building and present four novell hypotheses.

Our research protocol with more detailed discussions of our approach is available as appendix C. In the protocol, we also discuss in more detail why we consider our case study to be primarily an exploratory case study.

5.2.1 Selecting Cases

We searched for software developing organizations that are ...

- large (in terms of number of employees) because in large organizations the target group of potentially collaborating developers is bigger and potentially more "silos" (parts of the organization not collaborating at all) exist.
- established (quantified by the age of the organization) because extensive growth and changes typically found in young organizations might distort observations and conclusions drawn about IS practices and collaboration.
- large (in terms of revenue) to ensure they are economically relevant.

As a function of their size, all of our case organizations have development locations in multiple countries. However, we searched for organizations with the majority of development in a single country to minimize effects of globally distributed software development (like cultural mismatch, language barriers, ...).

We identified three organizations fitting these criteria. All three organizations brought us in as consultants and, by doing so, partially financed this research. Upon request of our contacts in the case organizations, we do not disclose the organizations' names. Instead, we refer to the organizations by using their primary domain as pseudonym: *automotive* org., *industry* org., and *medical* org.

Table 5.1 summarizes key attributes of each case organization. The selected organizations are similar in that they all have a large number of employees ($> 45,000$), are established organizations, and have globally distributed software development. In each of the organizations, a majority of employees in the same country.

Table 5.1: Overview of case organizations

	Automotive Org.	**Industry Org.**	**Medical Org.**
Domain	Automotive supplier developing hardware, software, and hardware-software components	Multi-industry organization catering to different domains including energy systems and factory automation.	Medical technology organization developing devices for diagnostic imaging and therapy support.
Annual Revenue (2017)	$>$ 40 billion EUR	$>$ 60 billion EUR	$>$ 10 billion EUR
Age of organization	$>$ 20 years	$>$ 20 years	$>$ 20 years
Number of employees	$> 200,000$	$> 300,000$	$> 45,000$
Regulation	Selected products regulated	Selected products regulated	Practically all products regulated
Development distribution	Globally distributed, but majority of development in same country	Globally distributed, but majority of development in same country	Globally distributed, but majority of development on same campus

However, the organizations differ on some dimensions. Medical org. is smaller (number of employees, revenue) than the other organizations and practically all of its products are regulated by government bodies (like the United States Food and Drug Administration). Industry org. caters to multiple domains while the other organizations are focused on a single domain.

5.2.2 Identifying IS Practices (Qualitative)

To understand how IS programs are set up in the studied organizations, we collected qualitative data. Subsequently, we performed a thematic analysis (Braun and Clarke 2006) of the data to identify collaboration practices. For each identified collaboration practice, we assessed whether it is an IS practice or not using the findings of prior literature.

Definitions

We define the following terms:

- A *practice* is a customary or expected procedure of performing a task or solving a problem.
- A (software development) *collaboration practice* is a practice used by an individual or group to collaborate with other individuals or groups (in the context of a software development effort).
- An *IS practice* is a collaboration practice from an open source (OS) context used for collaboration within an organization.
- A *non-IS practice* is any practice that is not an IS practice.

This thesis is in the field of software engineering. Thus, when talking about collaboration practices, we will not refer to them as *software development* collaboration practices because this is implied in the context.

Data Collection

We collected the qualitative case study data using direct observations, interviews, and retrieval of preexisting documentation and artifacts used by the organizations. Table 5.2 summarizes

Table 5.2: Considered qualitative data items

	Automotive Org.	**Industry Org.**	**Medical Org.**
Observations	5 Field notes ($A_{N1}, \ldots, A_{N5}$)	2 Field notes (I_{N1}, I_{N2})	8 Field notes ($M_{N1}, \ldots, M_{N8}$), 2 Email threads (M_{E1}, M_{E2})
Interviews	4 Semi-struct. ($A_{I1}, \ldots, A_{I4}$)	1 Unstructured (I_{I1}), 6 Semi-structured ($I_{I2}, \ldots, I_{I7}$)	1 Unstructured (M_{I1}), 2 Semi-structured (M_{I2}, M_{I3})
Documents	9 Process documents ($A_{D1}, \ldots, A_{D9}$)	3 Int. presentations ($I_{D1}, \ldots, I_{D3}$), 1 Internal wiki page (I_{D4})	2 Int. presentations (M_{D1}, M_{D2})
Artifacts	1 Internal leaflet on IS (A_{A1}), 9 Screenshots ($A_{A2}, \ldots, A_{A10}$)		

the considered qualitative data. In total, we considered 28 data items for automotive org., 13 for industry org., and 15 for medical org.

Observations. We performed direct observations in all the organizations by observing and participating in meetings and workshops. We documented the observation by taking manual field notes. At medical org., we also considered two email threads.

Interviews. We performed unstructured (I_{I1}, M_{I1}) and semi-structured (all others) interviews. A majority of interviews were not solely focused on collaboration or IS practices, but on more specific topics (e.g. goals of the organizations IS program, quality assurance). We performed audio recordings and transcriptions for the interviews. Before each recorded interview, we established informed consent with the interviewees (Singer and Vinson 2002). Interviewees were typically suggested by the organization and involved with IS either as contributors or by planning or being responsible for an IS program.

For two interviews (I_{I1}, M_{I1}), no audio recording was done and we took manual notes instead. When quoting those interviews in this thesis, we paraphrased from our notes. All interviews and field notes were in a language other than English. When quoting the interviews in this thesis, we translated them to English.

Other data. In addition, we collected documentation from all three organizations. For automotive org., we collected artifacts (a leaflet and screenshots from their software forge).

Data Analysis

We analyzed the data using a three phase approach.

Phase I) Thematic analysis. We performed a thematic analysis to identify which collaboration practices were exercised in the organizations' IS programs. As suggested by Braun and Clarke (2006), we followed a five step approach:

- We transcribed and familiarized ourselves with the data (step 1).
- We performed an initial coding (step 2). We labeled statements in the text with so called codes (short descriptions). We used the software tool MaxQDA and maintained a code book (Guest et al. 2006) listing definitions for each code.
- We grouped the codes into common themes (step 3). Each resulting theme, represents one observed practice.
- We reviewed whether the resulting codes fit the labeled segments and whether the codes and themes do justice to the entire data set (step 4).
- We then redefined and named themes and adapted our code book accordingly (step 5).

We iterated over these steps multiple times.

Phase II) Additional grouping. Our thematic analysis resulted in a total of 14 themes. We interpreted each theme to be one collaboration practice. We manually grouped the practices into four groups to make them easier to present, comprehend and compare.

Phase III) Assessment. We utilized existing IS literature to assess for each observed collaboration practice whether it is suggested by prior research work on IS (and thus an IS practice) or not mentioned or even discouraged by prior research work (and thus a non-IS practice).

5.2.3 Measuring Patch-Flow (Quantitative)

We measured the patch-flow in the case organizations and analyzed its magnitude and structure.

Definition of Scope

Before measuring the patch-flow one must decide which IS projects to include in the scope for measurement. We were interested in all software development efforts where IS collaboration was suspected. With the help of employees of the studied organization, we identified a total of five IS programs: one in industry org. (test infrastructure), two in each automotive org. (forge components, AutoSource) and medical org. (development tools, imaging platform). Each of the IS programs serves as one unit of analysis for our case study. For each unit of analysis, we measured the patch-flow for exactly one year. All IS programs were at least three months old at the first day included in the measurement interval.

Measurement

We measured the patch-flow using the method presented in chapter 3. We applied it as follows:

Step I) Extraction of code contributions. Using the patch-flow crawler (see chapter 4), we extracted the code contributions to IS projects at automotive org. (via the API of their GitHub forge) and medical org.'s development tools program (via the API of their Microsoft Team Foundation Server (TFS)). At industry org. and medical org.'s imaging platform, we utilized export scripts proprietary to each of these organizations.

Step II) Mapping of code contributions IS projects. We considered each repository in the forge to be a separate IS project (automotive org.) or utilized directory paths (all other organizations) to identify for each code contribution which IS project it was contributed to.

Step III) Extraction of organizational data. Using the patch-flow crawler, we extracted organizational data from the organizational databases used by the organizations (LDAP directory for automotive org., a collection of XML files for medical org.). For industry org., we did not get access to the employee database due to privacy concerns but an on-site engineering manager extracted and provided a model of the organizational structure to us.

Step IV) Identification of the author. We used the unique identifiers and email addresses of contributors (where available from the code contribution meta data) to identify the author for each code contribution with the help of the patch-flow crawler (automotive org., medical org.) Where such information was not available, we utilized an individuals name or department code to identify the individual manually with the help of on-site employees (automotive org., medical org.). Due to privacy concerns, an on-site employee performed these tasks at industry org. using the same process but not the patch-flow crawler.

Step V) Mapping of authors to org. units. Once we had identified the individuals and their records in the organizational databases (last step), we mapped them to their current org. units trivially by querying the respective organizational databases.

Step VI) Mapping of IS projects to org. units. Due to the large number of IS projects, we were not able to perform this step manually (e.g. by asking employees of the organizations which org. units are responsible for each IS project). Instead, we mapped each IS project to the org. unit whose developers contributed the most code contributions to it.

Exclusion of Data

For automotive org., we identified and excluded from analysis 49 repositories that contained OS projects imported into the organization's software forge, 119 repositories that were created as part of a Git training for employees, and one project in which an employee used the software forge to store automatically performed backups. In addition, we excluded certain code contributions from analysis. Table 5.3 presents the total number of code contributions excluded per case with the reason for exclusion.

Normalization of Organizational Data

Researchers can consider multiple dimensions when modeling an organization for patch-flow measurement, for example the disciplinary (often called "solid line"), functional (often called "dotted line"), regional, or legal entity organization. In this chapter, we focus on the highest levels of the disciplinary organization:

Table 5.3: Number of excluded code contributions by reason

Reason for exclusion	**Automotive Org.**	**Industry Org.**	**Medical Org.**
Branching or merging commits	(Excluded at time of measurement)		
Author not identified	9736	53	133
Author's org. not identified	5151	13	259
Receiving IS project not identified	0	709	35
Performed by bot, automated commits	(Excluded at time of measurement)		

- For automotive org., we consider their top level (divisions) and second level (business units) org. units.
- For industry org., we consider top level (segments), second level (business units), and third level org. units.
- For medical org., we consider top level (business areas) and second level (business lines) org. units.

Organizations differ from one another. As a consequence, we need an approach to establish comparability between organizations' org. units. We use the size of an org. unit (measured in number of total employees) as a base for comparison.

Automotive org. has over 200,000 employees, industry org. over 300,000 employees, and medical org. over 40,000 employees. The divisions (top level) of automotive org. have median over 45,000 employees; the segments (top level) of industry org. have median over 40,000 employees.

Based on this information, we normalize the levels of the org. units in the tree to put those org. units (or organizations) with 45,000 employees on the same level. Table 5.4 summarizes the normalized organizational levels (org. levels) and their corresponding native levels.

Uncovering Relationships Between IS Practices and Patch-Flow

To identify how IS practices affect IS collaboration (proxied by patch-flow), we seek to uncover and interpret correlations between IS practices and patch-flow observed in each IS program as well as co-existing patterns in the programs.

Table 5.4: Normalized organizational levels

Normalized Level	Automotive Org.	Industry Org.	Medical Org.
Level 1	Top level (Divisions)	Top level (Segments)	
Level 2	Second level (Business units)	Second level (Business units)	Top level (Business areas)
Level 3		Third level	Second level (Business lines)

Our research setup does not allow us to simply employ statistical correlation coefficients to identify such correlations: First, with five IS programs, our sample size is significantly too small to employ typical correlations like Pearson, Kendall, and Spearman correlations (Bonett and Wright 2000). Second, in case study research such cross-case (or cross-unit of analysis) syntheses should not rely solely "on numeric tallies" but rather "strongly on argumentative interpretation" (Yin 2013).

As a consequence, we provided an argumentative interpretation of our case study by providing a theory composed of four hypotheses theorizing about the relationship between IS practices and IS collaboration and grounding it in our case study observations.

5.3 Results: Case Descriptions

This section summarizes the context of each case organization and gives a rich description of each case. We present which practices are used for collaboration (both IS practices and non-IS practices) and the patch-flow for each unit of analysis. Table 5.5 sorts each of the studied IS programs and its IS projects into the classification framework presented in chapter 2.

5.3.1 Automotive Org. - AutoSource

Automotive org. is a large organization developing and supplying automotive parts and equipment to original equipment manufacturers and end customers. Multiple of its products are combination of proprietary software and hardware. In 2017, its yearly revenue was over 40 billion EUR. It has over 200,000 employees distributed globally but the majority of its research and development activities are performed within the same country.

Table 5.5: Classification of studied inner source programs

	Automotive Org.		**Industry Org.**	**Medical Org.**	
	AutoSource	Forge components	Test infrastructure	Imaging platform	Development tools
IS program					
Prevalence	• Selective	• Selective	• Selective	• Project-specific	• Selective
Degree of self-org.	• Free component choice • Assigned tasks	• Free component choice • Assigned tasks	• Free component choice • Assigned tasks	• Assigned components • Assigned tasks	• Free component choice • Assigned tasks
Market mechanisms	• Local library	• Local library	• Local library	• Elements of an internal market	• Local library
IS projects					
Governance	• Single org. unit	• Single org. unit	• Single org. unit	• Single org. unit	• Single org. unit
Objective	• Exploration-oriented • Utility-oriented	• Exploration-oriented • Utility-oriented	• Utility-oriented • Service-oriented	• Service-oriented	• Exploration-oriented • Utility-oriented

Automotive org. runs an IS program modeled after an OS foundation. The IS program's projects are hosted on an instance of the software forge GitHub Enterprise. We refer to this program as AutoSource.

The program was initiated by a central research and development division with the explicit mission to enable IS collaboration within automotive org. and enable collaboration and (re)use across silos within automotive org. It is a selective and local-library (Lindman et al. 2013) IS program which allows developers free choice of components but not tasks.

Collaboration practices

We identified nine collaboration practices that were exercised as part of industry org.'s test infrastructure program. These practices are the themes resulting from the thematic analysis of the collected qualitative data (phase I and II in section 5.2.2).

Open code for all parties to read, (re)use. Every employee within the organization is allowed to contribute to the IS projects in the AutoSource program:

> "The user has read access to all resources of [AutoSource]" [A_{D3}]

> "We designed [AutoSource] and every employee of [automotive org.] can participate. Its public and everybody can see what we do – everyone else within the company." [A_{I2}]

An internal marketing leaflet advertises AutoSource as enabling employees to "browse trough all projects in [AutoSource]" and to "download and use all [AutoSource] projects" [A_{A1}].

However, at the time of collecting our case study data, employees from two of automotive org.'s divisions were excluded from participating in AutoSource because their IT infrastructure is not compatible to that of the other divisions:

> "There are – but that is a technical problem – excluded parties [...]. They use another technical infrastructure than the remaining group." [A_{I4}]

We still consider this IS program to be open to all relevant parties because traditionally those two divisions develop less software, operate more independently, and are significantly smaller than the remaining divisions.

Provide single entrypoint to IS program. Automotive org. uses a software forge that provides single entrypoint to the IS program. The feature to "browse through all projects in [AutoSource]" is actively advertised to developers [A_{A1}]. We explored the listing [A_{A9}] and search feature [A_{A10}] of the software forge. In addition to the software forge, an AutoSource portal provides additional features for listing and finding IS projects.

Open code for contributions by all parties. Similar to reading the source code and (re)using the components, all parties within the organization are granted the right to contribute code contributions to the IS projects. The same technical incompatibilities like for (re)using components apply.

Establish committer role. Every code contribution towards an IS project in the AutoSource program has to be refereed and either integrated or rejected by a committer. Changes to the code base cannot be made without a committer's approval [A_{D4}, A_{D8}]. The AutoSource documentation defines the role of a committer (similar to our definition) as follows: "A committer is a contributor that was given write access to the code repository of a dedicated project" [A_{D3}].

Provide open dedicated communication infrastructure. In addition to the software forge, the AutoSource program offers its projects "user mailing lists, user-support forums" [A_{D3}]. Hyperlinks to the support forums and mailing lists are integrated into the portal [A_{A2}, A_{A10}].

Communicate using closed communication. Despite the availability of infrastructure for open communication, the employees of automotive org. exercised significant closed communication (that is communication that is not open communication).

Establish governance committees (inspired by OS foundations). AutoSource is governed by multiple committees:

> "[The AutoSource] operational board ensures daily work in [AutoSource] [...] [The AutoSource] governance board" steers [AutoSource] [...] The operational board should have members present in the governance board in order to act as glue between development and management and to bring operative sigh to the discussions]" [A_{D3}]

Each IS projects is governed by a project management committee (PMC) [A_{D3}].

Design & enforce IS license. An IS license explicitly governs the rights and obligations of internal parties (re)using the IS software components [A_D2, A_D3, A_N3, A_N4]. The license is modeled after an OS license. All IS projects in AutoSource need to be provided under this specific IS license. Contributors have to sign a contributor license agreement in which they grant others in the organization the right to use their contributions in accordance with the IS license: "The potential contributor has to agree that the software he products is under the AutoSource license" [A_{D2}].

Select & coach IS projects using project incubator. AutoSource runs a project incubator. "The incubator filters projects on the basis of the likeliness to become successful meritocratic communities" [A_{D3}], it coaches and "helps projects to start in [AutoSource]" [A_{D3}].

Patch-flow

Figure 5.1 shows the patch-flow (including all patches to projects of the AutoSource program) as a chord diagram with hierarchical edge bundling (Holten 2006). The white curved boxes represent org. units and the stacking of the boxes indicates the organizational hierarchy. We name org. units on level 1 with Roman numbers, level 2 with letters, and level 3 with Arabic numerals. Three points ("...") indicate that zero or more other org units exists, but did not contribute. We included functional org. units (for example departments for IT or corporate research) only if they were contributing and marked them with the keyword "func". Curved edges indicate the patch-flow from green (contributor) to red (receiver). The width of an edge indicates its weight (number of patches).

Table 5.6 summarizes selected metrics related to the program's patch-flow. The first two rows show how many of the program's IS projects receive contributions from at least two org. units (and, thus, at least one outside patch). The following rows show the number of org. units involved in patch-flow and the absolute and relative patch-flow. Relative patch-flow is the percentage of all code contributions being patches. The columns indicate the different org. levels considered (e.g. the bottom right cell shows the relative patch-flow across org. level 1).

The AutoSource program is an IS program of small size (45 IS projects) with moderate development activity (2493 code contributions in the considered year). Patch-flow occurs to measurable magnitude in the program (59 patches, 2.37% relative patch-flow) and a sizable fraction

Figure 5.1: Patch-flow in automotive org.'s AutoSource program

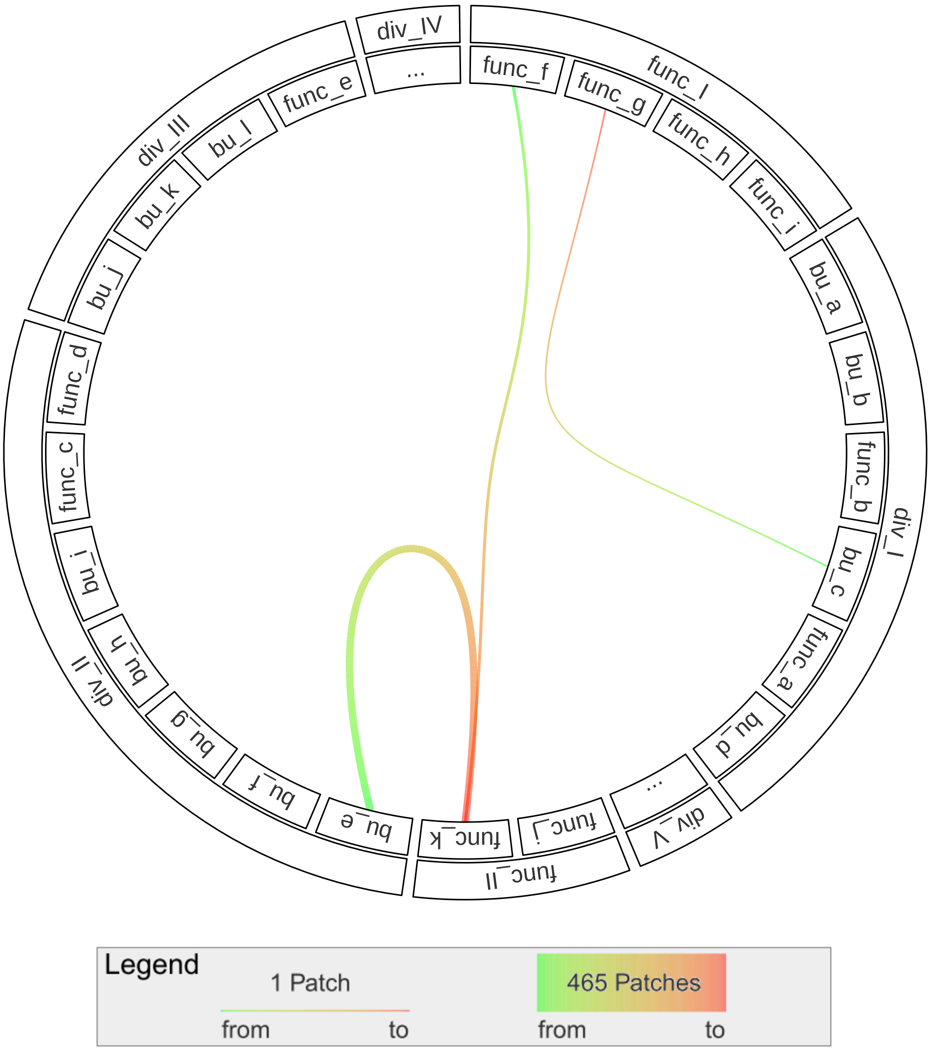


Table 5.6: Patch-flow metrics for automotive org.'s AutoSource program

	Level 3	Level 2	Level 1
IS projects with ≥ two org. units contributing			
Abs. (number of projects)	/	6 / 45	6 / 45
Rel. (percentage of projects)	/	13.33%	13.33%
Org. units involved in IS collaboration			
Abs. (number of org. units)	/	5	4
Patch-flow			
Abs. (number of code contributions)	/	59 / 2493	59 / 2493
Rel. (percentage of all code contributions)	/	2.37%	2.37%

of IS projects (six IS projects, 13.33%) receive outside contributions from other org. units on level 1 and 2. While only a few business units on level 2 (five) are involved in the collaboration, all except of one divisions on level 1 (four) are involved (when not counting the two org. units that are included in the program for technical reasons). All observed patch-flow relationships cross the highest level of automotive org. Three of the five business units and two of three the divisions involved in collaboration are functional org. units.

5.3.2 Automotive Org. - Forge Components

In addition to AutoSource, we found a second IS program at automotive org. The second IS program at automotive org. is more informal. Because before mentioned software forge exists, developers started their own IS projects on the forge independently of the AutoSource program. As these projects in this program are a lose collection of software components hosted on the software forge, we refer to them (and the informal IS program they form) as the forge components. It is a selective and local-library (Lindman et al. 2013) IS program which allows developers free choice of components but not tasks.

Collaboration practices

We observed five collaboration practices that were exercised for the forge components. The forge components are not part of a formal program but rather the projects were initiated in a self-organized fashion. The practices that are exercised were not designed intentionally by employees of automotive org. but are shaped by the used software forge and its features.

Open code for all parties to read, (re)use; open code for contributions by all parties. Similar to AutoSource, all parties within the organization can access the software forge, read software code, and (re)use and contribute to the IS projects. Similarly to AutoSource projects, the IS projects that are public within the organization [A_{N5}]. An interviewee summarized:

> "Well, allright, this is not only present in [AutoSource]. But [there is] collaboration where you have a shared repository, where everybody sees what others are doing there, and you self-organize" [A_{I2}].

Similarly to the AutoSource program, two divisions are excluded due to incompatibility of their infrastructure.

Provide single entrypoint to IS program. Automotive org.'s GitHub Enterprise forge serves as a single entrypoint to the forge components. Individuals searching for a specific IS project can make use of the features the forge provides, e.g. show a list of all recently created IS projects [A_{A9}].

Establish committer role. No processes or roles are defined explicitly. However, the used software forge implicitly enforces that not every developer has write access to IS project's repositories by default. Rather, write access must be granted explicitly. Thus, those who have write access to an IS projects must elevate others to committers by granting them write access for their project's repository [A_{A9}].

Provide dedicated open communication infrastructure. The used software forge provides an issue tracker for each IS project. As one specific issue tracker is automatically provided per IS project, we consider this dedicated open communication infrastructure.

Patch-flow

Figure 5.2 and table 5.7 summarize the patch-flow data for the forge component program. They follows the same conventions as the previous figure and table.

The forge components form a large IS program (1789 IS projects) and receive a large number of code contributions (37,368) in the considered time interval. A large number of org. units (17 on level 2) are involved in the collaboration. All divisions (except those excluded from the program for technical reasons) are involved in collaboration. However, only a small fraction of IS projects receives outside patches at all (67 or 3.75% across level 2; 40 or 2.24% across level 1). The absolute patch-flow is high (823 across level 2; 412 across level 1) but significantly lower on level 1 than on level 2. Despite the high absolute patch-flow, only a small fraction of code contributions are patches (2.20% across level 2, 1.10% across level 1).

5.3.3 Industry Org. - Test Infrastructure

Industry org. is a large organization catering to multiple domains in light and heavy industries. Its domains include energy systems and factory automation. In 2017 its yearly revenue was over 60 billion EUR. It has over 300,000 employees distributed globally but the majority of

Figure 5.2: Patch-flow in automotive org.'s forge components program

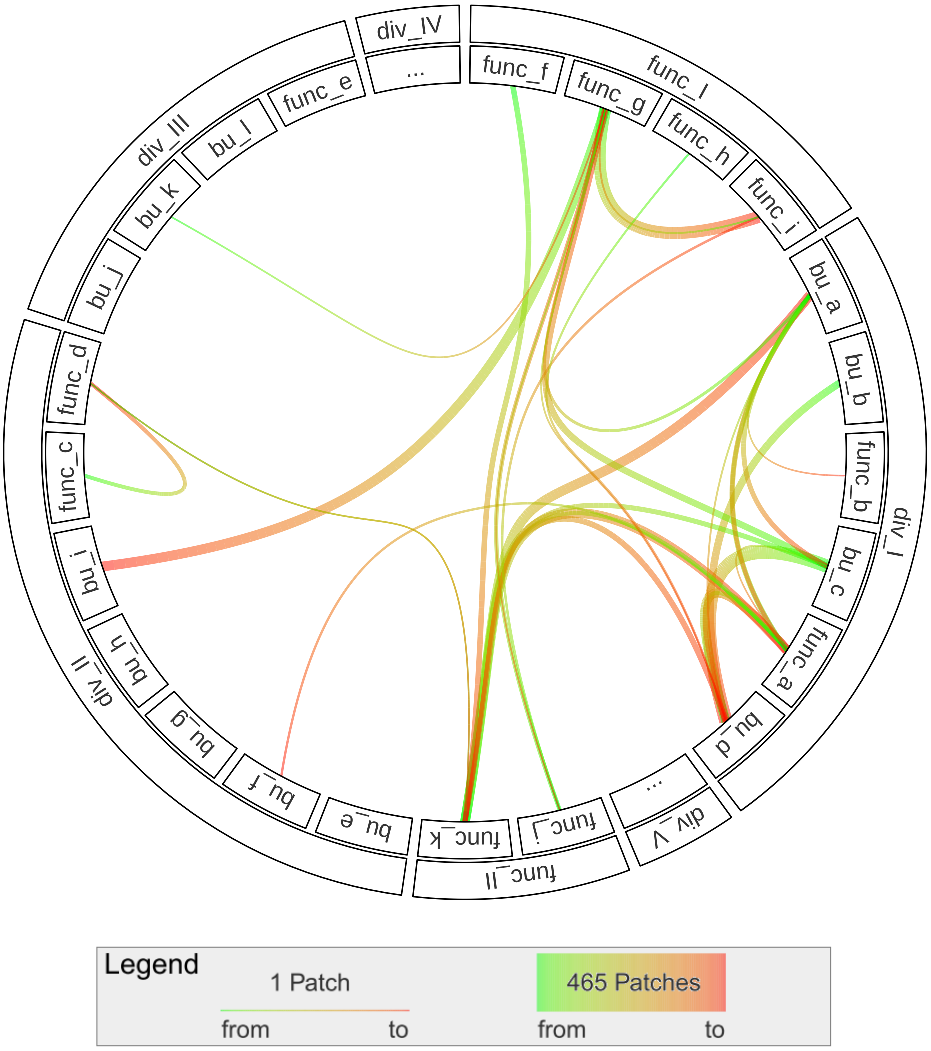


Table 5.7: Patch-flow metrics for automotive org.'s forge components program

	Level 3	Level 2	Level 1
IS projects with ≥ two org. units contributing			
Abs. (number of projects)	/	67 / 1789	40 / 1789
Rel. (percentage of projects)	/	3.75%	2.24%
Org. units involved in IS collaboration			
Abs. (number of org. units)	/	17	5
Patch-flow			
Abs. (number of code contributions)	/	823 / 37,368	412 / 37,368
Rel. (percentage of all code contributions)	/	2.20%	1.10%

its research and development activities are performed within the same country. Multiple of industry org.'s products consist of both proprietarily hardware and software.

Industry org. runs a program where developers of different teams can collaborate on components of their software test infrastructure and test tools. We refer to it as the test infrastructure program.

The program was initiated by an org. unit responsible for innovation and platform topics of one of industry org.'s flagship products. Dedicated teams are responsible for testing the components and feature clusters of this product. The program was initiated with the goal to avoid duplication of functionality in and increase reuse of test infrastructure and test tools. Before the initiation of the test infrastructure program, each of the teams would independently develop infrastructure components to test those components or features they are responsible for. The program is a selective and local-library (Lindman et al. 2013) IS program which allows developers free choice of components but not tasks.

Collaboration practices

We identified six collaboration practices that were exercised as part of industry org.'s test infrastructure program.

Open code for all parties to read and (re)use. All parties within industry org. can see the source code and (re)use the software components that are developed as part of the projects of the test infrastructure program:

> "The [...] code we develop: It's for everyone. We just have the responsibility for it." [I_{I2}]

> "Those [projects] are libraries – ready made functionality – that everybody can use. Not only our [team], but everybody. [...] I have read access. Also other teams can look at my source code. That's what it [the program] is made for" [I_{I3}]

> "I have full access on the source code [of the test infrastructure projects]" [I_{I4}]

Provide single entrypoint to IS program. To enable individuals to find components that are relevant to them, the team responsible for innovation and platform topics (discussed earlier) maintains a wiki page [I_{D5}] with information on each available project, contact persons, and links to further resources. The wiki page serves as a single entrypoint towards the IS program.

Open code for contributions by all parties. Every employee of the organization can contribute code to the program's projects:

> "Everybody can influence [a project] at any point in time: either reporting a bug or contributing an extension" [I_{I3}]

Employees also claimed to make use of the opportunity to contribute code to IS projects:

> "We fix bugs in [component a], [component b], [component c] ourselves as well." [I_{I5}]

> Referring to an IS project: "There is surely one or the other thing that we contributed because it was simply not yet available or because we have a solution already within our environment." [I_{I2}]

Perform ad hoc code review and mentoring. There is no committer role (as defined in chapter 1). Code reviews of contributions before their integration into the code base are not mandatory:

> "It's not mandatory that I have to do a review for each code contribution. Not mandatory. No." [I_{I3}]

> "There is nobody who examines the code, one checks in. You got to know what you are doing. And well, you check it in [into the repository] and then it's in" [I_{I5}]

> "In principle every body can check in [changes]. If it builds, it's in." [I_{I4}]

Even though not mandatory, code reviews are performed incidentally in an ad hoc fashion:

> "We at the [specific] team, we have the habit of at least asking those responsible for an [IS project] to perform a review." [I_{I2}]

While code review is not mandatory and no committer role is formally established, there are individuals that are responsible or considered responsible by their peers for specific IS projects. There are often the contact persons listed on the wiki page discussed above:

> "If we do our own work on a [specific IS project], then we do not just check in our contribution, but we write to [a specific employee], who is responsible for [the specific IS project] and ask him: 'We had to adapt something. Can you have a look at that?'" [I_{I2}]

Communicate using closed communication. A majority of communication in the organization and regarding the IS projects is done using synchronous and asynchronous closed communication. Interviewees responded as follows when asked how they communicate about their work on the IS projects:

> "Typically via communicator [an instant messaging tool], email, sometimes using face to face meetings. If somebody works in the same building, you get together." [I_{I5}]

> "They communicate among themselves, using communicator, email." [I_{I4}]

Establish informal governance mechanisms. Industry org. established no formal governance mechanisms (like standing committees) to govern the test infrastructure program. However, they established informal governance primarily by a set of recurring meetings:

> "There are regular status meetings for developers [of test infrastructure components and tools]" [I_{I5}]

> "[There is] a [test infrastructure] planning meeting. What's going on? What's the status? How to continue regarding tooling? For example. So, there are different things put on the agenda." [I_{I3}]

Patch-flow

Figure 5.3 and table 5.8 summarize the patch-flow data for the test infrastructure program. They follows the same conventions as the previous figure and table.

The test infrastructure program is of small size (35 IS projects) with low to moderate development activity (1853 code contributions). The majority of collaboration in the test infrastructure program happens within one hot spot: Between the level 3 org. units 1, 2, 3. These org. units are in close proximity to one another as they are both children of the same business unit and segment. There is high patch-flow (341 patches) between them. A few patches (seven) were contributed by org. units in higher distance (crossing level 2 and 1). A sizable fraction of IS projects (13 or 37.14% considering level 3; 4 or 11.43% considering level 2 and 1) received at least one outside patch.

5.3.4 Medical Org. - Imaging Platform

Medical org. is a hardware and software company producing medical products. Its main business focus is on integrated hardware-software products for diagnostic imaging (for example magnetic resonance imaging devices or computer tomography scanners). With 45,000 employees and an annual revenue of 10 billion EUR, medical org. is smaller than the other case organizations. As typical for medical products, practically all of medical org.'s products are tightly regulated by government bodies (like the United States Food and Drug Administration).

A majority of medical org.'s medical imaging devices are developed as part of a software product line (SPL) for medical imaging devices. We refer to the SPL's platform as the imaging platform. Inspired by IS, selected developers from one specific business unit developing products based on the platform (product unit) can perform code contributions to the imaging platform.

Figure 5.3: Patch-flow in industry org.'s test infrastructure program

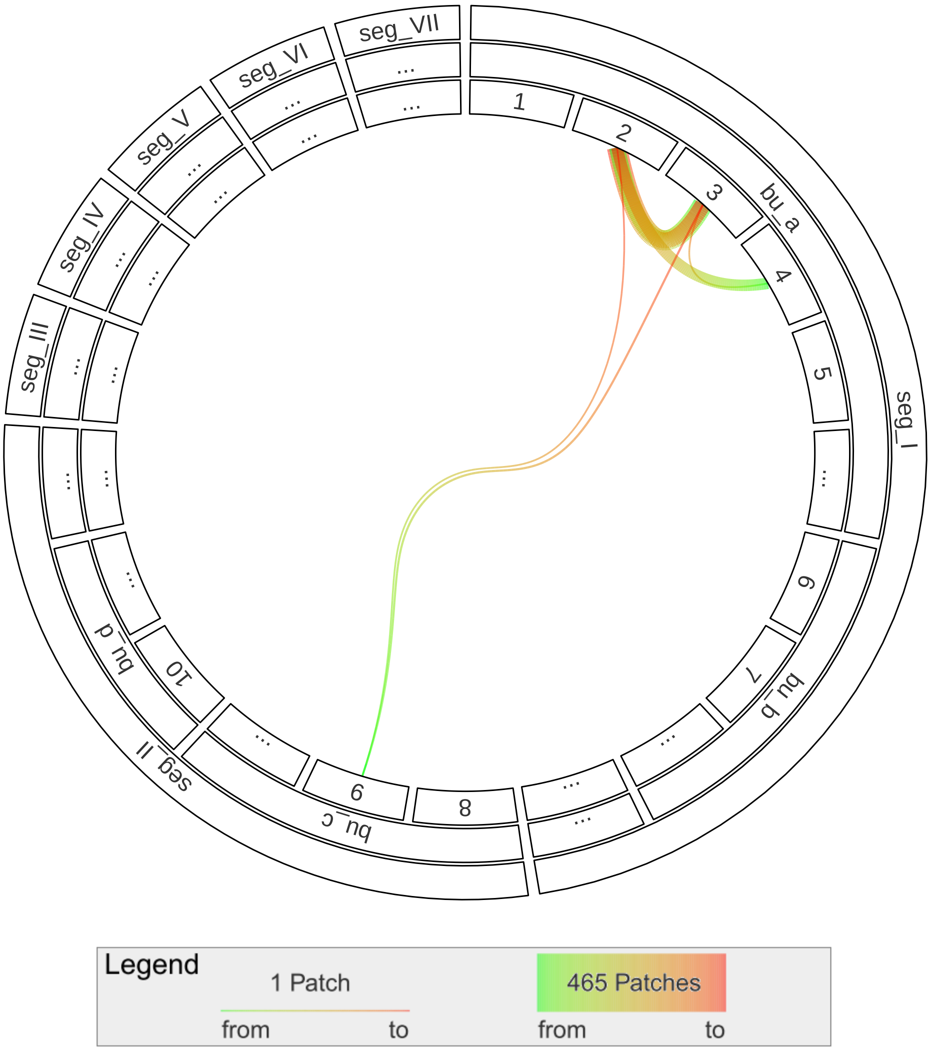


Table 5.8: Patch-flow metrics for industry org.'s test infrastructure program

	Level 3	Level 2	Level 1
IS projects with ≥ two org. units contributing			
Abs. (number of projects)	13 / 35	4 / 35	4 / 35
Rel. (percentage of projects)	37.14%	11.43%	11.43%
Org. units involved in IS collaboration			
Abs. (number of org. units)	4	2	2
Patch-flow			
Abs. (number of code contributions)	348 / 1853	7 / 1853	7 / 1853
Rel. (percentage of all code contributions)	18.78%	0.38%	0.38%

The imaging platform is a project-specific (Gurbani et al. 2010) IS program which allows developers free choice of components but not tasks. Partially, the product units indirectly finance the development of the platform by paying a fee to the platform unit. Thus, one could argue it shows elements of an internal-market (Lindman et al. 2013).

Collaboration practices

We identified four collaboration practices that were exercised as part of medical org.'s imaging platform program.

Open code to read, (re)use only for selected parties. All parties in the organization are granted read access to the source code [M_{I_3}]. However, the software components of the imaging platform cannot be freely reused by all developers within the organization. As typical in SPL engineering, only developers of org. units taking part in the software product line are allowed to use the software as part of their products [M_{I_3}].

Open code for contributions only by selected parties. The openness to contribute code contributions to the platform is restricted. As a consequence of regulatory requirements, every developer must receive training courses and a certification for the development processes he or she follows. Developers of a product unit are trained and certified in the processes of their product unit. Developers of the platform unit are trained and certified in the processes of the platform unit.

If a developer from a product unit wants to contribute to code from the platform unit (primarily the imaging platform), has to receive a specific training as well:

> "If I am a developer of [a product unit], and I want to contribute to the [platform], I can't just readily do it. [...] We designed a special delta-training with which a [product-]developer receives the necessary privileges – from the process perspective – by means of an additional training to change things at the [platform]." [M_{I_3}]

This training is only available to developers of one specific product unit [M_{D_1}]. As a consequence, only developers of this specific product unit are enabled to contribute code to the platform and only after they passed the training and received certification.

Establish mandatory code review (but not committer role). Every code contribution to the imaging platform has to be reviewed by an additional developer before it is integrated into the code repository:

> "So, mandated by our software process we have the obligation to review. Both handing in the code for review and performing the code review [...] has to follow specific guidelines." [M_{I3}]

> "Reviews of various artifacts are mandatory. [...] Product code must be reviewed by at least on additional software developer" [M_{I1}]

There is no dedicated committer, but contributors can select themselves who they ask for a review [M_{I1}, M_{I3}, M_{N6}]:

> "The developer can pick the reviewer himself [...] Often you simply ask somebody from your own team" [M_{I1}]

> "If a colleague doesn't react to a review ticket within two hours, the author will typically search for another reviewer" [M_{I1}]

Communicate using closed communication. We found no evidence of open communication being exercised as part of the IS collaboration on the imaging platform. Closed communication is exercised, for example in the form of face-to-face meetings [M_{I3}].

Patch-flow

Figure 5.4 and table 5.9 summarize the patch-flow data for the test infrastructure program. They follows the same conventions as the previous figure and table.

As a project-specific IS program (Gurbani et al. 2010, chapter 2), the IS program contains only one IS project – the imaging platform. However, this project is developed actively (20,395 code contributions in the considered year).

The chord diagram is dominated by one patch-flow edge with a high weight: The majority of patch-flow (464 patches) flow from the product unit bl_1 (whose developers are able to receive training and certification for the platform processes) to the platform unit bl_4. We observed one patch from the product unit bl_3. This is a consequence of developers switching business lines but still retaining their right to contribute to the platform. All org. units participating in collaboration are children of the same business area ba_b and are in close proximity to one another.

Figure 5.4: Patch-flow in medical org.'s imaging platform program

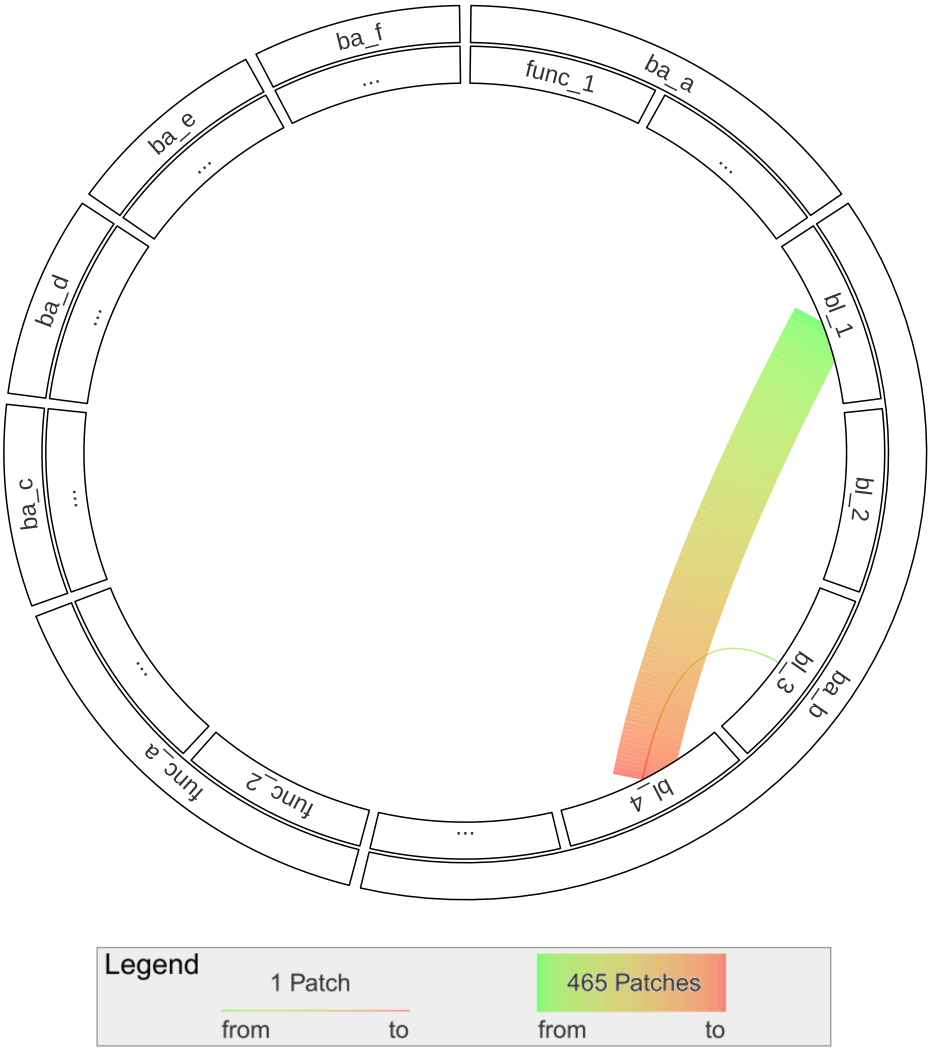


Table 5.9: Patch-flow metrics for medical org.'s imaging platform program

	Level 3	Level 2	Level 1
IS projects with $\geq$ two org. units contributing			
Abs. (number of projects)	1 / 1	0 / 1	/
Rel. (percentage of projects)	100.00%	0.00%	/
Org. units involved in IS collaboration			
Abs. (number of org. units)	3	1	/
Patch-flow			
Abs. (number of code contributions)	465 / 20,395	0 / 20,395	/
Rel. (percentage of all code contributions)	2.28%	0.00%	/

5.3.5 Medical Org. - Development tools

In addition to the imaging platform, medical org. runs the development tools IS program. The development tools program allows developers to provide and collaborate on non-product software (primarily development tools). The program does not use a software forge. Rather, a central TFS repository and additional intranet pages are provided. The program is a selective and local-library IS program (Lindman et al. 2013) which allows developers free choice of components but not tasks.

Collaboration practices

We found two collaboration practices to be exercised in the development tools program. We found no evidence of any practices for program governance or open communication.

Open code for all parties to read, (re)use. Internal marketing slides advertise the IS projects part of the development tools program to be "shared across [business lines]" [M_{D2}]. The repository that holds the development tools is accessible for all developers within the organization (even for us as researchers with very basic user privileges in the organization the repository was accessible and we could read the code).

Open code for contributions by all parties. Similarly, the repository showed no indication and we found no evidence of any restrictions of which developers can contribute to it. Upon request our contacts in medical org. confirmed to us that the development tools are open for code contributions by all parties.

Perform ad hoc code review and mentoring. We did not find any evidence suggesting that code review was generally mandatory for contributions to the development tools or even a committer role had been established. However, our contacts at medical org. confirmed to us that (similar to ad hoc code reviews observed at industry org.) some contributors may choose to ask a knowledgeable person or even somebody who is (perceived to be) responsible for an IS projects to review their code before it's contributed and there are no enforced rules regarding the review of code contributions.

Figure 5.5: Patch-flow in medical org.'s development tools program

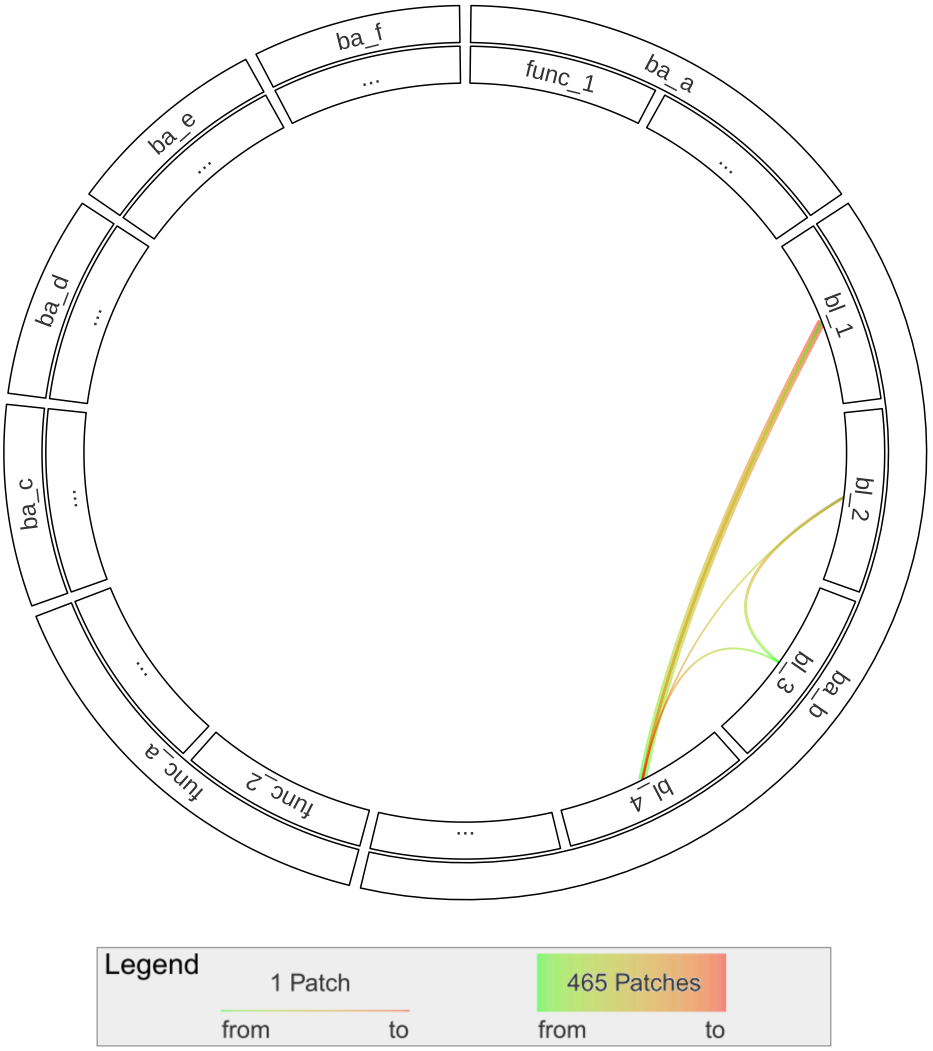


Table 5.10: Patch-flow metrics for medical org.'s development tools program

	Level 3	Level 2	Level 1
IS projects with $\geq$ two org. units contributing			
Abs. (number of projects)	7 / 120	0 / 120	/
Rel. (percentage of projects)	5.83%	0.00%	/
Org. units involved in IS collaboration			
Abs. (number of org. units)	4	1	/
Patch-flow			
Abs. (number of code contributions)	85 / 1511	0 / 1511	/
Rel. (percentage of all code contributions)	5.63%	0.00%	/

Patch-flow

Figure 5.5 and table 5.10 summarize the patch-flow data for the development tools program. They follows the same conventions as the previous figure and table.

The development tools are an IS program of moderate size (120 IS projects) with low development activity (1511 code contributions in the considered year). Only a small fraction of IS projects (7 or 5.85% considering level 3) receive any outside patches. Only few parties (four business lines) are involved in collaboration.

We observed only collaboration in close proximity between business lines on level 3 that are children of the same business area ba_b. We observed low to moderate patch-flow (85 patches). Due to the low development activity, the relative patch-flow moderate (5.63%).

5.4 Results: Cross Synthesis

The previous section provided a description of each IS program (unit of analysis) observed in the studied organizations (cases).

This sections synthesizes across the studied units of analysis. We compare the collaboration practices exercised in the IS programs and assess whether these are IS practices or non-IS practices (section 5.4.1), compare the observed patch-flow and lay out similarities and differences among the IS programs (section 5.4.2), and subsequently discuss correlations between the exercised IS practices and observed IS collaboration (section 5.4.3).

5.4.1 Inner Source Practices

Table 5.11 summarizes all practices we observed in the IS programs. These practices are the themes resulting from the thematic analysis of the collected qualitative data (resulting from phase I described in section 5.2.2). Over all studied organizations and IS programs, we identified a total of 14 IS practices. We give each practice a number from 1 to 14 and refer to the practice using this number throughout this chapter. The right column of the table indicates which data items the practices were derived from.

The gray bars separate the groups the practices belong to (resulting from the grouping in phase II). We grouped the IS practices into four groups (*a) Read, (re)use code*, *b) Contributions to code*, *c) Open communication*, and *d) Governance*).

Table 5.11: Summary of observed practices

Name	IS Practice?	Description	Data items
a) Read, (re)use code			
1) Open code for all parties to read, (re)use	✓ Yes	The source code of IS practices is opened for all developers within the organization and all developers can (re)use IS components.	A_{A1}, A_{D3}, A_{I2}, A_{I4}, I_{D5}, $I_{I2} - I_{I5}$, M_{I3}, M_{D2}
2) Provide single entrypoint to IS program	✓ Yes	A single entrypoint to the IS program and its project portfolio is provided (for example using a wiki or by a software forge with features to list and search for IS projects).	A_{A1}, A_{A9}, A_{A10}, I_{D1}, I_{D5}, I_{I2}
3) Open code to read, (re)use only for selected parties	✗ No	Only developers from selected org. units can read source code of IS projects or (re)use IS components.	M_{I3}
b) Contributions to code			
4) Open code for contributions by all parties	✓ Yes	Every developer in the organization can contribute code changes to the IS projects.	A_{D1}, A_{D3}, A_{I2}, I_{I3}, I_{I4}, I_{I5}, M_{N2}
5) Establish committer role	✓ Yes	Every IS project has at least one committer. Committers review code contributions and decide on their rejection or inclusion.	A_{A9}, A_{D3}, A_{D4}, A_{D8}
6) Perform ad hoc code reviews and mentoring	✗ No	Code review of code contributions before integration is not mandatory. In an ad hoc fashion, contributors incidentally ask a colleague with perceived knowledge or who is responsible for an IS project to review their patch.	$I_{I2} - I_{I5}$
7) Establish mandatory code review (but not committer role)	✗ No	Code review is mandatory before a code contribution is integrated into the code base. However, there is no dedicated committer but developers choose themselves whom to ask to review their code contribution.	M_{I1}, M_{I3}, M_{N6}

Table 5.11: Summary of observed practices - continued

Name	IS Practice?	Description	Data items
8) Open code for contributions only by selected parties	✘ No	Only developers from selected org. units or with specific credentials can contribute code changes to the IS projects. Other developers are excluded from contributing.	M_{N3}, M_{N4}
c) Open communication			
9) Provide dedicated open communication infrastructure	✔ Yes	The IS program provides dedicated infrastructure for open communication (for example mailing lists or forums).	A_{A2}, A_{A10}, A_{D3}
10) Communicate using closed communication	✘ No	The developers communicate regarding the IS projects using closed communication (communication that is not open communication) for example in the form of meetings, email, or telephone calls.	A_{D3}, A_{I1}, A_{I2}, $I_{I2} - I_{I5}, M_{E2}$
d) Governance			
11) Establish governance committees (inspired by OS foundations)	✔ Yes	The IS program is governed by governance committees inspired by those found at open source foundations (for example a steering committee governing the IS program, or a project management committee governing a specific IS project).	A_{D3}, A_{I1}, A_{N1}
12) Design & enforce IS license	✔ Yes	Each project has to be provided under a specific IS license that explicitly governs rights and obligations of internal parties using the software. Contributors have to sign a contributor license agreement.	A_{D2}, A_{D3}, A_{D5}, A_{N3}, A_{N4}
13) Select & coach IS projects using project incubator	✔ Yes	The IS program runs a project incubator with the goal to identify and coach new promising IS projects.	A_{D3}, A_{D6}, A_{D7}, $A_{N4}, A_{I1} - A_{I3}$
14) Establish informal governance mechanisms	Unclear	The IS program is governed using informal structures like recurring coordination meetings but no formal committees.	$I_{I2} - I_{I5}, I_{I7}$

Assessment of practices

For each collaboration practice resulting from the thematic analysis of the qualitative data, we used literature to assess whether it is an IS practice or a not (phase III in section 5.2.2).

We assess the practices as follows:

- IS is "open to all developers behind the firewall [of an organization]" (Dinkelacker et al. 2002) and gives these developers "universal access to [IS] development artifacts" (Stol et al. 2014). Prior work suggests inclusion of all parties (*1) Open code for all parties to read, (re)use; 4) Open code for contributions by all parties*) and discourages exclusion (*3) Open code to read, (re)use only for selected parties; 8) Open code for contributions only by selected parties*).

- Riehle et al. (2009) suggest to provide a software forge that serves as a starting point for finding IS projects (*2) Provide single entrypoint to IS program*). Software forges are widely used by practitioners (Cooper and Stol 2018).

- Typically, patches in IS need approval by a committer (Gurbani et al. 2006; Riehle et al. 2009; Cooper and Stol 2018) (*5) Establish committer role*). Not implementing a committer role is discouraged (*6) Perform ad hoc code reviews and mentoring; 7) Establish mandatory code review (but not committer role)*).

- Open communication is a key element of OS (Riehle 2015) and IS (Capraro and Riehle 2017). We consider practices encouraged if they support open communication (*9) Provide dedicated open communication infrastructure*) and discouraged if not (*10) Communicate using closed communication*).

- Governance practices that are inspired by OS (*12) Design & enforce IS license*) or OS foundations (*11) Establish governance committees (inspired by OS foundations); 13) Select & coach IS projects using project incubator*) are encouraged by prior work (Riehle 2016; Riehle et al. 2016).

We were not able to identify whether *14) Establish informal governance mechanisms* should be considered an IS or non-IS practice. On the one hand, there are hundreds of OS project who are governed in an informal manner. However, OS projects are typically not governed using recurring telephone meetings as observed at industry org.'s test infrastructure program.

In table 5.11, we marked the IS practices (those suggested by IS literature) with a green check mark symbol and non-IS practices (those discouraged by IS literature) with a yellow cross.

Practices across the IS programs

Table 5.12 shows which practices are implemented in each IS program. Similar to the previous table, we marked IS practices with a green check mark and non-IS practices with a yellow cross. The IS programs in the table are sorted left to right by ascending number of (green) IS practices.

Comparing the (IS and non-IS) practices across IS programs, we observed significant differences among the IS programs:

- There is significant diversity among the studied IS programs in terms of exercised IS practices. While there are programs that operate with no (medical org.'s imaging platform) or little (medical org.'s development tools) IS practices, other programs implement significantly more IS practices (automotive org.'s AutoSource). All IS programs exercised non-IS practices. Where IS practices where exercised, the non-IS practices were exercised in addition to the IS practices.

However, we also observed similarities among all (or a majority of) the studied IS programs:

- For none of the programs, we found that evidence of significant open communication being exercised (despite open communication being a key element of IS). Rather, we observed for three of the programs that significant closed communication (that is communication that is not open communication) takes place.

- A majority of programs (medical org.'s programs, automotive org.'s forge components) do not exercise any practices to govern the portfolio of IS projects or the program itself. Only the AutoSource program uses formal committees and mechanisms for governance. OS-inspired governance practices have low prevalence.

5.4.2 Patch-Flow

Table 5.13 summarizes patch-flow related metrics. Each column shows metrics for one IS program. The rows show the following: Rows in part (a) show an overview of the collected patch-

Table 5.12: Practices by inner source program

Practice	Medical Org.		Industry Org.	Automotive Org.	
	Imag.plat.	Dev.tools	Test infr.	For.comp.	AutoSource
a) Read & (re)use code					
1) Open code for all parties to read, (re)use		✓	✓	✓	✓
2) Provide single entrypoint to IS program			✓	✓	✓
3) Open code to read, (re)use only for selected parties	✗				
b) Contributions to code					
4) Open code for contributions by all parties		✓	✓	✓	✓
5) Establish committer role				✓	✓
6) Perform ad hoc code reviews and mentoring		✗	✗		
7) Establish mandatory code review (but not committer)	✗				
8) Open code for contributions only by selected parties	✗				
c) Open communication					
9) Provide dedicated open communication infrastructure				✓	✓
10) Communicate using closed communication	✗		✗		✗
d) Governance					
11) Establish governance committees (insp. by OS found.)					✓
12) Design & enforce IS license					✓
13) Select & coach IS projects using project incubator					✓
14) Establish informal governance mechanisms			□		

□ Exercised practice ✓ Exercised IS practice ✗ Exercised Non-IS Practice

Table 5.13: Overview of patch-flow metrics per inner source program

	Medical Org.		Industry Org.	Automotive Org.	
	Imag.plat.	Dev.tools	Test infr.	For.comp.	AutoSource
a) Data overview					
Interval	Sept '15 - Aug '16	Aug '16 - Jul' 17	Jul '15 - Jun '16	Jun '17 - May '18	Jun '17 - May '18
Time if measurement	Jul - Sept '16	Aug '17	Jun - Jul '16	Jun '18	Jun '18
Number of code contr.	20,395	1511	1853	37,368	2493
b) Org. Units involved in IS collaboration					
Level 3	3	4	4	/	/
Level 2	None	None	2	17	5
Level 1	/	/	2	5	4
c) Distance of collaborating org. units					
Distance	Close proximity	Close proximity	Majority in close proximity	High distance & close proximity	High distance
d) IS projects with ≥ two org. units contributing					
Level 3	100.00% (1/1)	5.83% (7/120)	37.14% (13/35)	/	/
Level 2	0.00% (0/1)	0.00% (0/120)	11.43% (4/35)	3.75% (67/1789)	13.33% (6/45)
Level 1	/	/	11.43% (4/35)	2.24% (40/1789)	13.33% (6/45)
e) Relative patch-flow					
Across level 3	Low (2.28%)	Moderate (5.63%)	High (18.78%)	/	/
Across level 2	None (0.00%)	None (0.00%)	Very low (0.38%)	Low (2.20%)	Low (2.37%)
Across level 1	/	/	Very low (0.38%)	Low (1.10%)	Low (2.37%)
f) Absolute patch-flow					
Across level 3	High (465)	Moderate (85)	High (348)	/	/
Across level 2	None (0)	None (0)	Very low (7)	High (823)	Moderate (59)
Across level 1	/	/	Very low (7)	High (412)	Moderate (59)

flow data. Part (b) summarizes the number of org. units involved in patch-flow considering different org. levels. Part (c) summarizes the distance of org. units involved in collaboration per IS program. Part (d) shows the percentage and number of IS projects with more than one org. units contributing to it considering the different org. levels, as well as the total number of IS projects in a program. Part (e) and (f) summarize the relative and absolute patch-flow per considering different org. levels.

The observed IS programs are of different activity. The number of total code contributions per IS program range from 1511 to 37,668 code contributions in the considered year. The observed patch-flow and related metrics differ significantly among the IS programs as well:

- The size of the IS programs, measured by the number of IS projects, differs significantly. Automotive org.'s forge components are large (1789 IS projects) while other programs like automotive org.'s AutoSource (45 IS projects) or industry org.'s test infrastructure (35 IS projects) are smaller.

- The distance of org. units involved in collaboration differs as well. While for some programs (those at medical org.) only parties in close proximity collaborate, we observed collaboration between parties in high distance in the other IS programs.

However, we also observed similarities among all (or a majority of) the studied IS programs:

- While the observed patch-flow in all studied IS programs differed in terms of how many patches were flowing, how many parties were involved in collaboration, and how high the distance between collaborating parties was, we still observed patch-flow in all of the studied IS programs.

- We found that for all IS programs a majority of code contributions do not constitute patch-flow across the considered org. levels. The majority of code contributions are code contributions made by employees of the org. units that are responsible for an IS project.

- The majority of IS projects (in the four studied programs that are not project-specific IS programs) does not receive outside patches.

5.4.3 Correlations

We identified correlations between IS practices that are exercised in an IS program and the resulting patch-flow. These correlations are not statistical correlations (like those expressed with Pearson, Kendall, or Spearman coefficients). Rather, we observed and compared IS practices and patch-flow across the different IS program as a base for an argumentative interpretation. We made the following observations:

- We observed that a higher number of IS practices exercised in an IS program, does not correlate with more absolute patch-flow. However, the number of IS practices correlate with higher relative patch-flow (percentage of all code contributions being patches). We will discuss in section 5.5, why we believe this correlation not to be caused by causal relationship.
- We observed that patch-flow happened in all IS programs including those where no IS practices were exercised. We will discuss in section 5.5 our interpretation that IS collaboration is also possible with rudimentary IS-inspired practices.
- We observed that a higher number of IS practices, correlates with IS collaboration among org. units in higher distance in the organization. We will discuss in section 5.5, why we believe this correlation to be caused by a causal relationship.
- We observed IS programs with governance practices to be smaller, but to have less projects receiving no outside patches. We will discuss in section 5.5, why we believe this to be caused by a causal relationship.

5.5 Interpretation

In this section, we interpret the case study observations to answer our research questions. We interpret the magnitude of observed IS collaboration (RQ 4) and the effect of IS practices on IS collaboration (RQ 5).

5.5.1 Magnitude of Inner Source Collaboration

We found that patch-flow exists to measurable extent in all five case study organizations. To the best of our knowledge, this is the first empirical evidence of IS collaboration among large org.

units within organizations. IS collaboration (measured using patch-flow) is possible within software developing organizations.

We found that both relative and absolute patch-flow differed across the studied IS programs. However for all IS programs, a large majority of code contributions did not constitute patch-flow (from outside org. units). We interpret the low patch-flow to mean that IS collaboration exists but is not everyday routine for the developers in the studied organizations.

5.5.2 Effect of Inner Source Practices on Collaboration

We interpret the case study observations by providing a theory (consisting of four hypotheses) on the relationship between IS practices and IS collaboration.

Effect on Magnitude of Collaboration

To discuss the magnitude of IS collaboration, we look both at the absolute patch-flow (number of patches across organizational boundaries) and the relative patch-flow (percentage of code contributions being patches across organizational boundaries).

Absolute patch-flow. We observed that a higher number of IS practices exercised in an IS program, does not correlate with more absolute patch-flow. At medical org.'s imaging platform, we observed the highest patch-flow between org. units on level 3 but the least IS practices implemented. Similarly, automotive org.'s forge components, showed the highest patch-flow on level 1 and 2, despite more IS practices being exercised in the AutoSource program.

Relative patch-flow. For the relative patch-flow, we observed a correlation to the number of IS practices. However, we do not believe this to be caused by a causal relationship because it is influenced by other factors: Medical org.'s imaging platform is actively maintained and developed by developers who work on the platform full time. Thus, it received more (non-patch-flow) code contributions significantly decreasing the relative patch-flow. In addition, the difference in relative patch-flow between automotive org.'s forge components and AutoSource is not significant. We formulate the following hypothesis:

Hypothesis 1: A higher number of IS practices alone does not necessarily result in more IS collaboration.

Rather, we believe additional contextual factors influence how much IS collaboration is taking place.

Desire to Collaborate vs. Friction while Collaborating

Two of these contextual factors are the desire or need of an individual to contribute on the one hand, and (process) obstacles imposing friction when contributing on the other hand.

We observed a diverse set of IS practices implemented by the analyzed IS programs. We found programs with many IS practices (those at industry org. or automotive org.) but also a program where we identified not a single IS practice (medical org.'s imaging platform). Yet, we observed patch-flow (and thus IS collaboration) at all studied IS programs.

In the case of medical org.'s imaging platform, as typical for a large SPL, hundreds of developers' daily work is built upon medical org.'s imaging platform. If a product developer in an SPL engineering setup requires a change to the platform, this typically triggers a lengthy and costly process (Riehle et al. 2016). Diverting to an alternative component when changes are needed is not possible: Only one such platform is available to developers of medical org. We formulate the following hypothesis:

Hypothesis 2: Significant IS collaboration is possible with rudimentary IS practices (*high friction*) if there is a strong need or desire to collaborate.

Effect on Distance of Collaborating Org. Units

We observed the following correlation: Where more IS practices were implemented, the distance of collaborating org. units increased. In medical org.'s programs with no or few IS practices, we only observed patch-flow between org. units in close proximity. In the other IS programs, we observed more patch-flow across org. units in higher proximity.

We believe this correlation is the result of a causal relationship: Employees of org. units in close proximity often have preexisting social or work relationships to one another. These relationships can be the result of geographical co-location, shared second level managers, work in a similar domain or on the same product or product family, and other reasons. Org. units in high distance are less likely to have such preexisting relationships.

Because of this, we believe IS practices to benefit particularly collaboration among org. units in high distance:

They benefit from open communication infrastructure (practice 9) because they will otherwise be excluded from communication between those in closer proximity. Governance practices (practice 11, 12, 13) ensure similar rights and obligations to all parties involved in an IS project. This gives org. units in high distance the same voice as their directly neighboring org. units. Similarly, a single entry point to the IS program (practice 2) or mentoring provided by a committer (practice 5) aids those in high distance, who have little other possibilities to find a far away IS project or receive help through existing social or work relationships. This is a well-known effect in OS as well where parties in even higher organizational distance (i.e. parties that typically are not part of the same organization) collaborate. We formulate the following hypothesis:

> **Hypothesis 3:** IS practices enable collaboration among org. units in high organizational distance.

Effect of Governance Practices

The two programs that exercise governance practices (test infrastructure, AutoSource), host a comparatively low number of IS projects (35 and 45). However, a relatively high percentage of their projects receive outside patch-flow (for example 11.43% and 13.33% on level 2).

We believe this to be an effect of the exercised governance practices. First, the practice *13) Select & coach IS projects using project incubator* ensures that only those projects with a potential for later collaboration are included in an IS program. The coaching of projects makes sure they are fit to receive contributions. Second, governance practices make it non-trivial for individuals or teams to create and run a new IS project. As a result, we believe self-selection leads only those parties who are serious about initiating a new IS project to go through with it despite the cost of exercising governance practices (e.g. forming a PMC or participating in recurring governance telephone calls).

In contrast, those IS projects with fewer restrictions and no governance practices (development tools, forge components), have comparatively high numbers of projects (120, 1789) but a significantly lower percentage of these IS projects receive outside patch-flow (0.00% and 3.75% on level 2). Many IS projects might be created without much thought or because it is a convenient way for developers to receive a place to dump software source code of arbitrary quality and with mixed potential for collaboration. We formulate the following hypothesis:

Hypothesis 4: Governance practices and restrictions in project initiation contribute to a small IS program but with a higher percentage of projects attracting outside patches.

Being able to attract outside patch-flow is one indicator of an IS project's success.

5.6 Trustworthiness

Similar to the case study we presented in chapter 3 (evaluating the viability and utility of the patch-flow method), we discuss the trustworthiness (and included in that also the limitations) of this case study along the dimensions credibility, transferability, dependability, and confirmability proposed by Guba (1981).

5.6.1 Credibility

Credibility is the degree to which we can establish confidence in the truth of our findings in the context of the inquiry (Guba 1981). We undertook the following steps to increase the credibility of our case study findings.

We performed peer debriefing (Guba 1981). We intensively discussed this work and gathered feedback from colleagues on multiple occasions.

We performed member checks (Guba 1981). In all three organizations, we presented the findings from our analysis in shortened practitioner-focused reports. In addition, we constantly gathered feedback on our interpretations from employees of all three studied organizations. In each automotive org. and medical org. we performed an additional dedicated member check session with a knowledgeable employee focusing on the practices exercised in the IS programs.

We exercised prolonged engagement (Guba 1981) in all three cases. For each of the case organizations, the time between the first and last qualitative data item collected was over 15 months.

Sparse qualitative data regarding two IS programs. Of the qualitative data we collected, only a minority contained insights regarding automotive org.'s forge components and medical org.'s development tools. We believe this to not be a limitation of our research but rather an effect of the more informal nature of these programs. The tactics to increase credibility described above mitigate this limitation: The prolonged engagement allowed us to get a more comprehensible view and allowed us time to "check [our] own developing perceptions"

(Guba 1981). The additional member checks focused on only the exercised practices allowed to focus the study members' attention to potential faults resulting from the sparse qualitative data regarding these two IS programs.

Identification of open communication. We did not find evidence of significant open communication in the studied organizations or regarding the analyzed IS programs. We believe that open communication (at least to a small extent) could exist in the case study organizations and not be known to the interviewed employees and not covered in the documentation and artifacts we studied. We propose future research employing quantitative methods (e.g. measuring the non-code contribution flow) to identify open communication practices and the magnitude of open communication.

Correctness of organizational model. A fault in the extracted organizational model can significantly impact the resulting patch-flow. For each organization, we verified the correctness of the extracted organizational model using additional materials such as internal telephone directories, annual and quarterly reports, secondary organizational databases, or with the help of onsite employees.

Re-organizations over time. Like Guzzi et al. (2012), we observed that the organizational databases did not contain historical data. Because we only studied the top two levels (top three for industry org.), we were able to use annual and quarterly reports to verify that no changes in the high-level structure occurred within our measurement interval. From the time we started the patch-flow measurement, we looked back for no more than 13 months to minimize the effects of individual employees switching teams.

Focus on high org. levels. In itself, the focus on only the highest levels of the organization introduces a new limitation: patch-flow between lower level organizations cannot be measured. We believe this to not have a big impact on discussing the magnitude of patch-flow observed: As already discussed in chapter 3, we suspect patch-flow between lower level org. units to be more common. Thus, demonstrating that patch-flow can happen also across high levels of an organization is of more relevance. When interpreting the effect of IS practices on

IS collaboration, we explicitly discussed the which org. levels we considered when building our hypotheses.

5.6.2 Transferability

Transferability is the degree to which findings of our inquiries hold validity in other context (Guba 1981). For our case study, we selected large and established organizations. The setup of our case study does not allow us to draw immediate conclusions to a whole population of large and established organizations exercising IS practices. We suggest future research to validate whether our hypotheses apply to the whole population.

5.6.3 Dependability

Dependability is the degree of stability of the findings and traceability from collected data to the findings (Guba 1981). We established dependability by providing an audit trail (Guba 1981) linking each identified code and theme to the unmodified data in our case study database (Yin 2013) with the help of the software tool MaxQDA.

5.6.4 Confirmability

Confirmability is the degree to which we are neutral towards our inquiry and might bias the findings (Guba 1981). To address confirmability, we performed researcher triangulation. Two researchers took turns in collecting the data. One researcher was performing the thematic analysis, however the two remaining researchers provided continuous feedback on the results. In addition, we addressed confirmability using member checks and peer debriefing as described in paragraph on establishing credibility.

Case organizations as funding sources. The case organizations partially funded this research. We do not believe this to have any negative impact on our neutrality and thus the confirmability of our research. Our key contacts in the organization appeared to be driven by an honest interest to learn about the status of their IS collaboration. However, it did influence our sampling: As a consequence of our research partially being funded by the organizations, all organizations in the sample were organizations with a budget for adopting and running IS.

5.7 Conclusion

With this study, we provided the first in-depth case study on IS utilizing quantitative in additional to qualitative insights.

Patch-flow (and thus IS collaboration) existed in all IS programs we studied - indicating that OS-style collaboration is possible in organizations developing proprietary (non-OS) software. However, we found that only a fraction of code contributions to IS projects constitute patch-flow across the considered levels of the organization.

An increased number of IS practices correlate with more org. units in higher distance being involved in IS but not necessarily with more patches flowing. To us, this indicates that IS is capable to enable collaboration among org. units in far distance, where no collaboration is typically happening: Unlike established software development approaches, IS can scale software development efforts to the largest organizations.

6

Closing

This section closes the thesis by summarizing our results and discussing their consequences (section 6.1), proposing future research using, extending, or following the work presented in this thesis (section 6.2), and giving an outlook (section 6.3).

6.1 Results and Consequences

In this section, we summarize the answers to each of our five research questions and discuss the consequences of these answers.

6.1.1 RQ1: What are the elements of IS software development?

Answer. Four key elements constitute inner source (IS) and are in a relationship with one another. An *open environment* is created by opening up development artifacts, inviting external contributions, and establishing open communication. *Shared cultural values* are internalized by individuals within the organization. Empowered by the open environment and shared cultural values, *communities around software* form. Project-specific and program-wide communities collaborate in *specific IS scenarios*. Collaboration in such scenarios can be observed in open source (OS) as well.

Consequences. The identified elements of IS provide a comprehensive definition of what IS is. On the one hand, this creates clarity for practitioners who want to adopt IS. It reduces the risk of IS becoming a "buzzword". On the other hand for researchers who want to do research on IS and related phenomena, such a definition provides a solid foundation for their research work. Also, the elements can provide a lens for practitioners to plan their adoption and provide a foundation for further work to assess IS.

6.1.2 RQ2: How do different IS implementations differ from one another?

Answer. We found that both IS projects and IS programs differ from one another. IS programs differ on at least three dimensions (how *prevalent* IS can become in the organization, what *degree of self-organization* individuals are granted, and what the program's *internal economics* are). IS projects differ on at least two dimensions (i.e. how many parties exercise *ownership* over the projects and what its *objective* is). We proposed a classification framework that lays out the classes for each of the dimensions of IS programs and projects.

Consequences. Similar to the identification of elements that constitute IS, the classification framework can be considered base research that gives researchers are more solid foundation to conduct their research on IS. Particularly, it lays out variation points of IS programs and projects. In doing so, it provides researchers with vocabulary to discuss more precisely the transferability or generalizability of their findings.

The classification framework enables practitioners to decide and plan what classes of IS they need and want to adopt within their organization. This allows them to choose exactly the classes of IS that provide an optimum of cost and benefit to them.

6.1.3 RQ3: How to measure IS collaboration within a software developing organization?

Answer. IS collaboration can be measured by measuring the patch-flow as its proxy. Patch-flow is the flow of code contribution across organizational boundaries such as org. unit, project, or cost center boundaries. We presented the patch-flow method for measuring the patch-flow within an organization. The necessary flow data can be extracted automatically from repositories and data sources that typically exist in large organizations. We found patch-flow measurement using our method to be viable and the results useful to practitioners. It is possible to measure the flow of (non-code) contributions in analogy to patch-flow.

Consequences. The patch-flow method is the first of its kind to measure IS collaboration. It takes into account the structure of the organization and the organizational distance of collaboration org. units. It can serve as a base for additional metrics on IS. Founding such metrics on patch-flow allows to practitioners to easily reuse them in the context of multiple organizations.

In addition, it can serve as an operational definition of IS collaboration and thus help researchers to establish construct validity on what they consider IS or not. In addition, patch-flow (or other types of contribution flow) can serve as an operational definition of where IS collaboration occurs. Some organizations use the term IS to describe initiatives with no collaboration occurring, while other organizations do not use the term IS despite existing IS collaboration.

6.1.4 RQ4: What is the magnitude of IS collaboration in organizations?

Answer. Patch-flow (and thus IS collaboration) happens with measurable but small magnitude: Only a small fraction of code contributions constitute patch-flow crossing the highest levels of an organization.

Consequences. We presented the first empirical quantitative evidence of IS collaboration. This shows that OS-style collaboration within organizations is possible. This indicates the relevance of research into IS (to researchers) and adoption efforts (to practitioners).

The observation that a majority of contributions do not constitute patch-flow raises the question who (if not outside contributors) is performing this work. This indicates opportunities for future work to identify what the community structures of IS communities are and whether established models from the OS context apply to IS as well.

6.1.5 RQ5: How do IS practices affect IS collaboration?

Answer. We proposed a theory on the effect of IS practices on IS collaboration composed of four hypotheses. We found that higher number of IS practices alone does not necessarily result in more IS collaboration (hypothesis 1). Significant IS collaboration is possible with rudimentary IS practices if there is a strong need or desire to collaborate (hypothesis 2). IS practices enable collaboration among org. units in high organizational distance (hypothesis 3). Governance practices and restrictions in project initiation contribute to a small IS program but with a higher percentage of projects attracting outside patches (hypothesis 4).

Consequences. The hypothesis that more IS practices do not necessarily result in more IS collaboration has consequences for practitioners adopting and running IS: One cannot just establish seemingly perfect IS practices and expect IS collaboration. Rather, the hypothesis that more IS collaboration can occur with rudimentary practices if there is a need to collaborate, indicates that practitioners should actively identify and integrate into their IS program software components for which such a need or desire to collaborate exists. This could lead to more IS collaboration within the organization.

Practitioners benefit from governance practices if they seek to establish a smaller but higher quality IS program.

Our hypothesis that IS practices enable collaboration among parties in high organizational distance, indicates that IS practices might be particularly beneficial for practitioners seeking to establish collaboration among such parties. We theorize that IS scales well to large organizations.

6.2 Future Research

Based on the findings of this thesis, we identified multiple opportunities for future research. Primarily, we suggest future work to extend our IS taxonomy, use and build upon the patch-flow method, explore the flow of non-code contributions, to more systematically transfer open source findings, models, and theories to the IS context, and to investigate governance of IS programs more thoroughly.

6.2.1 Extend the IS Taxonomy

We presented an IS taxonomy consisting of a model of elements that constitute IS and a classification framework for IS projects and programs. We suggest future work to validate, extend, and build upon it.

Validation and extension of classification framework. Our classification framework for IS programs and projects is based on surveyed literature. We utilized it to describe the IS projects and programs in the case study presented in chapter 5. However, we suggest that future research should validate the classification framework (for example by surveying a larger and representative sample of organizations using IS). We believe that such research might also uncover additional dimensions and classes for classifying IS programs and projects. We encourage researchers to extend our classification frameworks.

Alternative IS scenarios. As part of the model of IS elements, we identified specific IS scenarios in which IS was exercised. We doubt their completeness. We believe many scenarios for collaboration can be observed in OS and within organizations and suggest future research to evaluate whether IS collaboration can take place in other scenarios and identify such scenarios.

Measurement instruments and assessment framework. The early phases of IS (IS adoption) has been extensively covered by the literature that fed into our IS taxonomy. However, the taxonomy does not yet consider how IS programs evolve. We suggest future work to develop an assessment or maturity framework for IS programs based on both our IS taxonomy and the contribution-flow phenomenon. Such work must include measurement instruments that allow primarily practitioners to locate their IS program within the resulting framework.

Evolution of IS programs and projects In this thesis, the evolution of IS programs and projects was not a significant lense: Our taxonomy helps to understand and classify the current state of an IS program. It is unclear how IS programs and IS projects evolve and which similarities they share with OS. We propose future research to investigate models describing and predicting the evolution of IS programs and projects.

6.2.2 Build upon Patch-Flow Method

We believe the insights from study presenting the patch-flow method and the case study applying the it suggest further research. In particular, we propose to develop management metrics for IS programs and projects and survey the collaboration in known IS programs.

Management metrics. Patch-flow measurement provides primitive data about the IS collaboration. We encourage researchers to use such data to theorize about and validate metrics for practitioners' information needs regarding IS (e.g. models for evaluating the performance of IS projects, incentive systems for IS).

Survey of inner source collaboration. In this thesis, we explored the structure and magnitude of IS collaboration (using patch-flow as a proxy). We suggest future research to validate these hypotheses and their transferability to other cases. This could for example be done by surveying a diverse and representative sample of IS programs, identifying their IS practices, and measuring the patch-flow there.

6.2.3 Explore Non-Code Contribution-Flow and Open Communication

Patch-flow analysis focuses on collaboration that happens directly on code. As we discussed in this thesis, non-code contribution is an important element of IS collaboration as well.

Non-code contribution flow. In addition to a code contribution, an individual may contribute to IS by reporting a bug, reviewing the contribution of somebody else, taking part in a mailing list discussion or by other means. We suggest future research to explore flow of non-code contributions as well. Further case study research could explore the magnitude of non-code contribution flow (in comparison to patch-flow) and the influences IS practices have on it.

We discussed that a limitation of the studies presented in this thesis is, that the prevalence of open communication might have been estimated to low because we were purely seeking to uncover instances of open communication using qualitative data analysis. Analyzing the non-code contribution-flow could mitigate this limitation.

Open communication mechanisms. Even considering this limitation, we had expected more open communication practices in the studied case organizations. This could suggest that established open communication mechanisms and tools from the OS context are not working well within organizations. We suggest future research to identify challenges to open communication within organizations and attributes of practices and tools that benefit adoption and acceptance of open communication within organizations.

6.2.4 Systematically Transfer Open Source Insights

Open source is in an inherent relationship with IS (we defined IS as the use of OS practices within organizations). We suggest further research into which models, methods, taxonomies – generally, which knowledge – from an OS context, can be applied within an IS context as well.

Community structure. For example, we observed that only a small fraction of all code contributions to an IS project constitute patch-flow across the highest levels of an organization. A potential explanation is that the Onion model (Nakakoji et al. 2002) describing the structure of OS communities, applies to IS communities as well. Following this explanation, a small core team of developers would perform the majority of work on the IS project and patch-flow then would most happen when individuals not in the core team contribution code. Recent papers have looked at the motivations of peripheral developers (for example Barcomb et al. 2019) and it is possible this research could be extended to an IS context.

Other similarities to open source. To generate additional insights about IS at relatively low cost, we encourage researchers to replicate studies that were performed in an OS context within an IS context. As a consequence, the research community can learn how similar OS and IS actually are, where differences between them lie and what the differences' consequences are. The practitioner community can benefit from learning just what OS practices and insights, still hold true in an IS context and adapting their IS programs accordingly – potentially leading to more successful IS collaboration.

6.2.5 Investigate Program Governance

In this thesis, we identified multiple topics regarding the governance of IS programs that warrant additional research.

IS license. In our multiple-case case study, we found an organization that established an IS license to make explicit the rights and obligations of organizational units and legal entities of the organization when using and contributing to IS software. We suggest further research to identify what elements an IS license should consist of, how it is different from the licenses used in an OS context, and how practitioners can define their own IS license.

Incentive system and career paths. Organizations have an interest in fostering IS collaboration (as it can lead to more efficient software development) and retaining talented software developers. We believe IS and the measurement methods this thesis presented can be used by human resource professionals to provide incentive systems well aligned with an organization's goals and career paths the reward individuals participating in IS collaboration. We suggest future research to develop and validate such incentive systems and career paths.

6.3 Outlook

The research presented in this thesis is basic research: During the literature survey (chapter 2), we identified a small but steady stream of scientific literature and practitioner reports on IS that are the foundation of our IS taxonomy. We developed a method to measure IS collaboration (chapter 3, chapter 4), applied it in a multiple-case case study (chapter 5) with three industry organizations and delivered the first in-depth quantitative study of IS collaboration. We found IS collaboration to happen in all studied organizations to low but clearly measurable magnitude. All this indicates to us that IS is still a young research and practitioner discipline. While our work contributes to a more solid foundation for IS, the future research section above clearly shows that many questions still remain unanswered. *Thus, we encourage other researchers to look into the IS phenomenon from the perspective of their fields.*

Such further research will be worthwhile. The results of this thesis indicate that IS can have a significant impact on the software industry: Our taxonomy (chapter 2) and the diversity of IS practices observed in our multiple-case case study (chapter 5) show that there are multiple ways to design and run IS programs successfully. As a consequence, the IS approach fits a diverse (and thus potentially large) set of organizations. In addition, we found that IS practices enable collaboration among parties in high distance within an organization (chapter 5). In that, IS is similar to its cousin OS and delivers something, other established software development approaches cannot deliver: *IS scales software development efforts to the largest organizations.*

References

Gary Anthes. 2005. Software Reuse: Making it Work - DTE Energy may have cracked the cultural side of reusaable software. (2005). http://www.computerworld.com/article/2556383/app-development/software-reuse---making-it-work.html

Matt Asay. 2007. Microsoft Office experiments with open source (development). (2007). http://archive.oreilly.com/pub/post/microsoft{_}office{_}experiments{_}w.html

Ann Barcomb, Andreas Kaufmann, Dirk Riehle, Klaas-Jan Stol, and Brian Fitzgerald. 2019. Uncovering the Periphery: A Qualitative Survey of Episodic Volunteering in Free/Libre and Open Source Software Communities. *IEEE Transactions on Software Engineering* (2019), 1–1. https://doi.org/10.1109/TSE.2018.2872713

Jörg Bartholdt and Detlef Becker. 2012. Scope extension of an existing product line. In *Proceedings of the 16th International Software Product Line Conference on - SPLC '12 -volume 1 (SPLC '12)*. ACM Press, New York, New York, USA, 275. https://doi.org/10.1145/2362536.2362573

K Beck. 1999. Embracing change with extreme programming. *Computer* 32, 10 (oct 1999), 70–77. https://doi.org/10.1109/2.796139

Andrew Begel, Khoo Yit Phang, and Thomas Zimmermann. 2010. Codebook: discovering and exploiting relationships in software repositories. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - ICSE '10*, Vol. 1. ACM Press, New York, New York, USA, 125. https://doi.org/10.1145/1806799.1806821

Douglas G Bonett and Thomas A Wright. 2000. Sample size requirements for estimating pearson, kendall and spearman correlations. *Psychometrika* 65, 1 (mar 2000), 23–28. https://doi.org/10.1007/BF02294183

Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative Research in Psychology* 3, 2 (jan 2006), 77–101. https://doi.org/10.1191/1478088706qp063oa

Maximilian Capraro. 2013. *Towards a representative and diverse analysis of issue-tracker related code and process metrics*. Master Thesis. Friedrich-Alexander-University Erlangen-Nuernberg.

Maximilian Capraro, Michael Dorner, and Dirk Riehle. 2018. The patch-flow method for measuring inner source collaboration. In *Proceedings of the 15th International Conference on Mining Software Repositories - MSR '18*. IEEE, ACM Press, New York, New York, USA, 515–525. https://doi.org/10.1145/3196398.3196417

Maximilian Capraro and Dirk Riehle. 2017. Inner Source Definition, Benefits, and Challenges. *Comput. Surveys* 49, 4 (dec 2017), 1–36. https://doi.org/10.1145/2856821

Noel Carroll, Lorraine Morgan, and Kieran Conboy. 2018. Examining the Impact of Adopting Inner Source Software Practices. In *Proceedings of the 14th International Symposium on Open Collaboration - OpenSym '18*. ACM Press, New York, New York, USA, 1–7. https://doi.org/10.1145/3233391.3233530

Herbert H Clark and Susan E Brennan. 1991. Grounding in communication. American Psychological Association, 222–233.

Danese Cooper and Klaas-Jan Stol. 2018. *Adopting InnerSource: Principles and Case Studies*. O'Reilly Media, Inc.

Gabriella C B Costa, Francisco Santana, Andréa M Magdaleno, and Cláudia M L Werner. 2014. Monitoring Collaboration in Software Processes Using Social Networks. In *CYTED-RITOS International Workshop on Groupware*. Springer, 89–96. https://doi.org/10.1007/978-3-319-10166-8_8

Kevin Crowston and James Howison. 2005. The social structure of free and open source software development. (2005). https://firstmonday.org/article/view/1207/1127

Kevin Crowston, Kangning Wei, James Howison, and Andrea Wiggins. 2012. Free/Libre open-source software development: What we know and what we do not know. *Comput. Surveys* 44, 2 (feb 2012), 1–35. https://doi.org/10.1145/2089125.2089127

Daniela S Cruzes and Tore Dyba. 2011. Recommended Steps for Thematic Synthesis in Software Engineering. In *2011 International Symposium on Empirical Software Engineering and Measurement*. IEEE, IEEE, 275–284. https://doi.org/10.1109/ESEM.2011.36

Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. 2012. Social coding in GitHub: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work - CSCW '12 (CSCW '12)*. ACM Press, New York, New York, USA, 1277. https://doi.org/10.1145/2145204.2145396

Jamie Dinkelacker and P Garg. 2001. Corporate source: Applying open source concepts to a corporate environment (Position Paper). *1st Workshop on Open Source Software Engineering* (2001). https://www.hpl.hp.com/techreports/2001/HPL-2001-135.pdf

Jamie Dinkelacker, Pankaj K Garg, Rob Miller, and Dean Nelson. 2002. Progressive open source. In *Proceedings of the 24th international conference on Software engineering - ICSE '02*. ACM, ACM Press, New York, New York, USA, 177. https://doi.org/10.1145/581339.581363

Kathleen M. Eisenhardt. 1989. Building Theories from Case Study Research. *Academy of Management Review* 14, 4 (oct 1989), 532–550. https://doi.org/10.5465/amr.1989.4308385

Roy Fielding. 2000. *Architectural Styles and the Design of Network-based Software Architecture*. PhD Thesis. University of California, Irvine. https://www.ics.uci.edu/{~}fielding/pubs/dissertation/fielding{_}dissertation.pdf

Brian Fitzgerald. 2006. The Transformation of Open Source Software. *MIS Q.* 30, 3 (sep 2006), 587–598. http://dl.acm.org/citation.cfm?id=2017296.2017298

Martin Fowler. 1997a. *Analysis patterns: reusable object models*. Addison-Wesley Professional.

Martin Fowler. 1997b. Dealing with properties. (1997). https://martinfowler.com/apsupp/properties.pdf

Steve Fox. 2007. IBM Internal Open Source Bazaar. (2007).

Javier Franch Gutiérrez, Angelo Susi, Maria Carmela Annosi, Claudia Patricia Ayala Mart\'\inez, Ruediger Glott, Daniel Gross, Ron Kenett, Fabio Mancinelli, Pop Ramsany, Cedric Thomas, and Others. 2013. Managing risk in open source software adoption. In *ICSOFT 2013: Proceedings of the 8th International Joint Conference on Software Technologies*. 258–264.

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1994. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional.

Gary Gaughan, Brian Fitzgerald, Lorraine Morgan, and Maha Shaikh. 2007. An examination of the use of inner source in multinational corporations. In *1st OPAALS Workshop*.

Gary Gaughan, Brian Fitzgerald, and Maha Shaikh. 2009. An Examination of the Use of Open Source Software Processes as a Global Software Development Solution for Commercial Software Engineering. In *2009 35th Euromicro Conference on Software Engineering and Advanced Applications*. IEEE, 20–27. https://doi.org/10.1109/SEAA.2009.86

Ron Goldman and Richard P Gabriel. 2005. *Innovation happens elsewhere: open source as business strategy*. Morgan Kaufmann.

Google-Blog. 2006. Google's 20 percent time in action. (2006). http://googleblog.blogspot.de/2006/05/googles-20-percent-time-in-action.html

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 (MSR '08)*. ACM, New York, NY, USA, 129–132. https://doi.org/10.1145/1370750.1370781

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

Greg Guest, Arwen Bunce, and Laura Johnson. 2006. How Many Interviews Are Enough? An Experiment with Data Saturation and Variability. *Field Methods* 18, 1 (feb 2006), 59–82. https://doi.org/10.1177/1525822X05279903

Vijay K Gurbani, Anita Garvert, and James D Herbsleb. 2005. A Case Study of Open Source Tools and Practices in a Commercial Setting. In *Proceedings of the Fifth Workshop on Open Source Software Engineering (5-WOSSE)*. ACM, New York, NY, USA, 1–6. https://doi.org/10.1145/1082983.1083264

Vijay K Gurbani, Anita Garvert, and James D Herbsleb. 2006. A case study of a corporate open source development model. In *Proceeding of the 28th international conference on Software engineering - ICSE '06 (ICSE '06)*. ACM Press, New York, New York, USA, 472. https://doi.org/10.1145/1134285.1134352

Vijay K Gurbani, Anita Garvert, and James D Herbsleb. 2010. Managing a corporate open source software asset. *Commun. ACM* 53, 2 (feb 2010), 155. https://doi.org/10.1145/1646353.1646392

Anja Guzzi, Andrew Begel, Jessica K Miller, and Krishna Nareddy. 2012. Facilitating enterprise software developer communication with CARES. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*. IEEE, IEEE, 527–536. https://doi.org/10.1109/ICSM.2012.6405317

Gary Hamel. 2008. *The Future of Management*. Vol. 16. Harvard Business School Press. hrmid.2008.04416fae.001 pages. https://doi.org/10.1108/hrmid.2008.04416fae.001

Nikolay Harutyunyan. 2019. *Corporate Open Source Governance of Software Supply Chains*. Ph.D. Dissertation.

Nikolay Harutyunyan and Dirk Riehle. 2019. Getting started with open source governance and compliance in companies. In *Proceedings of the 15th International Symposium on Open Collaboration - OpenSym '19 (OpenSym '19)*. ACM Press, New York, New York, USA, 1–10. https://doi.org/10.1145/3306446.3340815

Constantin Hasler. 2017. *Implementierung und Performance-Optimierung von SCM Adaptern*. Master Thesis. Friedrich-Alexander-University Erlangen-Nuernberg.

Hillside-Group. 2010. How to Hold a Writers Workshop. (2010). http://hillside.net/conferences/plop/235-how-to-hold-a-writers-workshop

Danny Holten. 2006. Hierarchical Edge Bundles: Visualization of Adjacency Relations in Hierarchical Data. *IEEE Transactions on Visualization and Computer Graphics* 12, 5 (sep 2006), 741–748. https://doi.org/10.1109/TVCG.2006.147

Martin Höst, Klaas-Jan Stol, and Alma Oručević-Alagić. 2014. Inner Source Project Management. In *Software Project Management in a Changing World*, Günther Ruhe and Claes Wohlin (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 343–369. https://doi.org/10.1007/978-3-642-55035-5_14

Mitchell Joblin, Wolfgang Mauerer, Sven Apel, Janet Siegmund, and Dirk Riehle. 2015. From Developer Networks to Verified Communities: A Fine-Grained Approach. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. IEEE Press, IEEE, 563–573. https://doi.org/10.1109/ICSE.2015.73

Eirini Kalliamvakou, Georgios Gousios, Diomidis Spinellis, and Nancy Pouloudi. 2009. Measuring developer contribution from software repository data. *Proceedings of the 2009 mediterranean conference on information systems - MCIS '09* (2009). https://aisel.aisnet.org/mcis2009/55

P.B. Kruchten. 1995. The 4+1 View Model of architecture. *IEEE Software* 12, 6 (1995), 42–50. https://doi.org/10.1109/52.469759

Johan Linåker, Maria Krantz, and Martin Höst. 2014. On Infrastructure for Facilitation of Inner Source in Small Development Teams. In *Product-Focused Software Process Improvement*, Andreas Jedlitschka, Pasi Kuvaja, Marco Kuhrmann, Tomi Männistö, Jürgen Münch, and Mikko Raatikainen (Eds.). Lecture Notes in Computer Science, Vol. 8892. Springer International Publishing, 149–163. https://doi.org/10.1007/978-3-319-13835-0_11

Juho Lindman, Mikko Riepula, Matti Rossi, and Pentti Marttiin. 2013. Open Source Technology in Intra-Organisational Software Development—Private Markets or Local Libraries. In *Managing Open Innovation Technologies*, Jenny S Z Eriksson Lundström, Mikael Wiberg, Stefan Hrastinski, Mats Edenius, and Pär J Ågerfalk (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 107–121. https://doi.org/10.1007/978-3-642-31650-0_7

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, Boston, MA, 381–387. https://doi.org/10.1007/978-0-387-09684-1_36

Juho Lindman, Matti Rossi, and Pentti Marttiin. 2010. Open Source Technology Changes Intra-Organizational Systems Development-A Tale of Two Companies. In *18th European Conference on Information Systems*.

Garm Lucassen, Fabiano Dalpiaz, Jan Martijn E. M. van der Werf, and Sjaak Brinkkemper. 2016. The Use and Effectiveness of User Stories in Practice. In *International working conference on requirements engineering: Foundation for software quality*. Springer, 205–222. https://doi.org/10.1007/978-3-319-30282-9_14

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

Guy Martin and Andrew Aitken. 2012. Inner Sourcing - Community Development Practices in Corporate IT. (2012). https://www.slideshare.net/blackducksoftware/innersource-webinar-series

Guy Martin and Aaron Lippold. 2011. Forge.mil: A Case Study for Utilizing Open Source Methodologies Inside of Government. In *Open Source Systems: Grounding Research*, Scott A Hissam, Barbara Russo, Manoel G de Mendonca Neto, and Fabio Kon (Eds.). IFIP Advances in Information and Communication Technology, Vol. 365. Springer Berlin Heidelberg, 334–337. https://doi.org/10.1007/978-3-642-24418-6_28

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

Catharina Melian. 2007. *Progressive Open Source: The construction of a development project at Hewlett-Packard*. Ph.D. Dissertation.

Catharina Melian, Cathy Burles Ammirati, Pankaj Garg, and Guje Sevon. 2002. *Building Networks of Software Communities in a Large Corporation*. Technical Report HPL-2002-12. HP Laboratories Palo Alto. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.10.2987{&}rep=rep1{&}type=pdf

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

Andrew Meneely and Laurie Williams. 2011. Socio-technical developer networks: should we trust our measurements?. In *Proceeding of the 33rd international conference on Software engineering - ICSE '11*. ACM, ACM Press, New York, New York, USA, 281. https://doi.org/10.1145/1985793.1985832

Aron Metzig. 2019. *Implementation of a Gitlab Adapter and the Evolution if its Interface*. Master Thesis. Friedrich-Alexander-University Erlangen-Nuernberg.

Microsoft. 2008. Open Source at Microsoft - Bringing the Open Source Approach In-House. (2008).

Lorraine Morgan, Joseph Feller, and Patrick Finnegan. 2011. Exploring inner source as a form of intraorganisational open innovation. In *Proceedings of the 19th European Conference on Information Systems*.

Kumiyo Nakakoji, Yasuhiro Yamamoto, Yoshiyuki Nishinaka, Kouichi Kishida, and Yunwen Ye. 2002. Evolution patterns of open-source software systems and communities. In *Proceedings of the international workshop on Principles of software evolution - IWPSE '02 (IWPSE '02)*. ACM Press, New York, New York, USA, 76. https://doi.org/10.1145/512035.512055

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

Patrick Oor, René Krikhaar, and I C T NoviQ. 2008. Balancing Technology, Organization, and Process in Inner Source. *Dagstuhl Workshop 08142: Combining the advantages of product lines and open source* (2008), 1548.

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

Tim O'Reilly. 2000. Archived email discussion on Open Source and OpenGL. (2000). `https://web.archive.org/web/20170104150601/http://archive.oreilly.com/pub/a/oreilly/ask{_}tim/2000/opengl{_}1200.html`

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 - SIGSOFT '08/FSE-16*. ACM, ACM Press, New York, New York, USA, 2. `https://doi.org/10.1145/1453101.1453105`

Daniele S Plantera. 2018. *Measuring Patch-Flow at an Automotive Company*. Master Thesis. Friedrich-Alexander-University Erlangen-Nuernberg.

Eric Raymond. 1999. The cathedral and the bazaar. *Knowledge, Technology & Policy* 12, 3 (sep 1999), 23–49. `https://doi.org/10.1007/s12130-999-1026-0`

Ralf Reussner and Wilhelm Hasselbring. 2006. *Handbuch der Software-Architektur*. dpunkt-Verlag.

Dirk Riehle. 2000. *Framework design: A role modeling approach*. Ph.D. Dissertation.

Dirk Riehle. 2007. The Economic Motivation of Open Source Software: Stakeholder Perspectives. *Computer* 40, 4 (apr 2007), 25–32. `https://doi.org/10.1109/MC.2007.147`

Dirk Riehle. 2009. The Commercial Open Source Business Model. In *Value Creation in E-Business Management*, Matthew L Nelson, Michael J Shaw, and Troy J Strader (Eds.). Lecture Notes in Business Information Processing, Vol. 36. Springer Berlin Heidelberg, 18–30. `https://doi.org/10.1007/978-3-642-03132-8_2`

Dirk Riehle. 2012. The single-vendor commercial open course business model. *Information Systems and e-Business Management* 10, 1 (mar 2012), 5–17. `https://doi.org/10.1007/s10257-010-0149-x`

Dirk Riehle. 2015. The Five Stages of Open Source Volunteering. In *Crowdsourcing*, Wei Li, Michael Huhn, and Wei-Tek Tsai (Eds.). Springer, 25–38. https://doi.org/10.1007/978-3-662-47011-4_2

Dirk Riehle. 2016. *An Example Charter for Inner Source Programs*. Technical Report. Friedrich-Alexander-University Erlangen-Nürnberg.

Dirk Riehle, Maximilian Capraro, Detlef Kips, and Lars Horn. 2016. Inner Source in Platform-Based Product Engineering. *IEEE Transactions on Software Engineering* 42, 12 (dec 2016), 1162–1177. https://doi.org/10.1109/TSE.2016.2554553

Dirk Riehle, John Ellenberger, Tamir Menahem, Boris Mikhailovski, Yuri Natchetoi, Barak Naveh, and Thomas Odenwald. 2009. Open Collaboration within Corporations Using Software Forges. *IEEE Software* 26, 2 (mar 2009), 52–58. https://doi.org/10.1109/MS.2009.44

Jason E Robbins. 2007. Adopting Open Source Software Engineering (OSSE) Practices by Adopting OSSE Tools. In *Perspectives on Free and Open Source Software*, Joseph Feller, Brian Fitzgerald, Scott Hissam, and Karim Lakhani (Eds.). The MIT Press, 245–264. https://doi.org/10.7551/mitpress/5326.003.0019

Per Runeson, Martin Höst, Austen Rainer, and Björn Regnell. 2012. Case Study Research in Software Engineering. In *Guidelines and examples*. John Wiley & Sons, Inc., Hoboken, NJ, USA. https://doi.org/10.1002/9781118181034

Andreas Schreiber, Roberto Galoppini, Michael Meinel, and Tobias Schlauch. 2014. An Open Source Software Directory for Aeronautics and Space. In *Proceedings of The International Symposium on Open Collaboration - OpenSym '14*. ACM, ACM Press, New York, New York, USA, 1–7. https://doi.org/10.1145/2641580.2641630

Michael Schwind and Christian Wegmann. 2008. SVNNAT: Measuring Collaboration in Software Development Networks. In *2008 10th IEEE Conference on E-Commerce Technology and the Fifth IEEE Conference on Enterprise Computing, E-Commerce and E-Services*. IEEE, IEEE, 97–104. https://doi.org/10.1109/CECandEEE.2008.100

Anthony Senyard and Martin Michlmayr. 2004. How to Have a Successful Free Software Project. In *11th Asia-Pacific Software Engineering Conference*. IEEE, 84–91. https://doi.org/10.1109/APSEC.2004.58

Srinarayan Sharma, Vijayan Sugumaran, and Balaji Rajagopalan. 2002. A framework for creating hybrid-open source software communities. *Information Systems Journal* 12, 1 (jan 2002), 7–25. https://doi.org/10.1046/j.1365-2575.2002.00116.x

Janice Singer and N.G. Vinson. 2002. Ethical issues in empirical studies of software engineering. *IEEE Transactions on Software Engineering* 28, 12 (dec 2002), 1171–1180. https://doi.org/10.1109/TSE.2002.1158289

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

Andrew Stellman and Jennifer Greene. 2009. *Beautiful Teams: Inspiring and Cautionary Tales from Veteran Team Leaders*. O'Reilly Media, Inc.

Klaas-Jan Stol. 2011. *Supporting product development with software from the bazaar*. Ph.D. Dissertation.

Klaas-Jan Stol, Paris Avgeriou, Muhammad Ali Babar, Yan Lucas, and Brian Fitzgerald. 2014. Key factors for adopting inner source. *ACM Transactions on Software Engineering and Methodology* 23, 2 (apr 2014), 1–35. https://doi.org/10.1145/2533685

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 (dec 2011), 1319–1336. https://doi.org/10.1016/j.infsof.2011.06.007

Klaas-Jan Stol and Brian Fitzgerald. 2014. *Research protocol for a case study of crowdsourcing software development*. Technical Report. University of Limerick, Lero.

Klaas-Jan Stol and Brian Fitzgerald. 2015. Inner Source–Adopting Open Source Development Practices in Organizations: A Tutorial. *IEEE Software* 32, 4 (jul 2015), 60–67. https://doi.org/10.1109/MS.2014.77

David R Thomas. 2006. A General Inductive Approach for Analyzing Qualitative Evaluation Data. *American Journal of Evaluation* 27, 2 (jun 2006), 237–246. https://doi.org/10.1177/1098214005283748

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 (jan 2011), 88–122. https://doi.org/10.17705/1jais.00252

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 - MSR 2014*. ACM, ACM Press, New York, New York, USA, 304–307. https://doi.org/10.1145/2597073.2597093

Frank van der Linden. 2009. Applying open source software principles in product lines. *Cepis Upgrade - The european journal for the informatics professional* 10 (2009), 32–41.

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

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

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 (oct 2010), 277–304. https://doi.org/10.2753/MIS0742-1222270209

Jane Webster and Richard T Watson. 2002. Analyzing the past to prepare for the future: Writing a literature review. *MIS quarterly* 26, 2 (2002), xiii—-xxiii.

Jacco Wesselius. 2008. The Bazaar inside the Cathedral: Business Models for Internal Markets. *IEEE Software* 25, 3 (may 2008), 60–66. https://doi.org/10.1109/MS.2008.79

James A Whittaker, Jason Arbon, and Jeff Carollo. 2012. *How Google tests software*. Addison-Wesley.

Claes Wohlin. 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering - EASE '14*. Citeseer, ACM Press, New York, New York, USA, 1–10. `https://doi.org/10.1145/2601248.2601268`

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

This is version *2078b873415cf78aaea60fc8e555df6613d14477* generated on Friday 5th June, 2020.

A

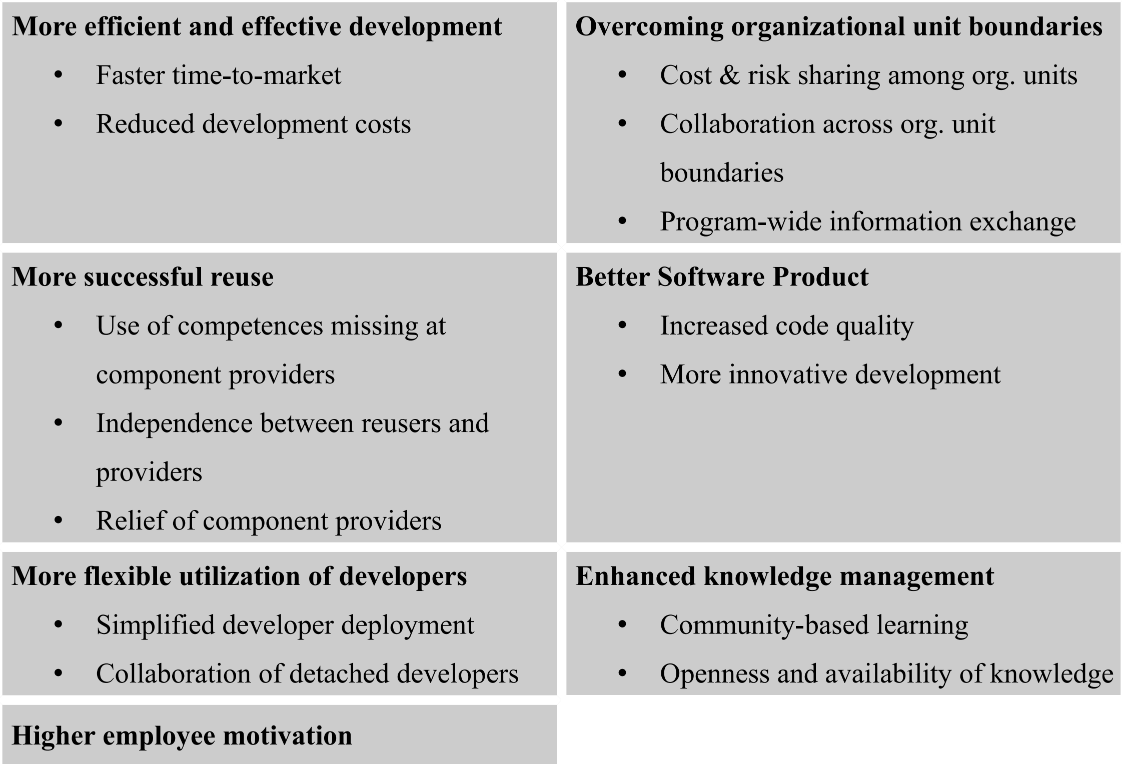
Claimed Benefits of Inner Source Adoption

This section summarizes the benefits practitioners observed or expect from inner source (IS) adoption. We used the literature selected as part of the literature survey study presented in chapter 2 and followed the approach presented by Thomas (2006) to identify the benefits. A detailed description of the approach and related work can be found in Capraro and Riehle (2017).

The surveyed literature presented IS benefits from the idiosyncratic perspective of different organizations. Some of the reports validated the observed IS benefits. Van der Linden et al. (2009) monitored Philips' process metrics to measure a time-to-market increase they attribute to inner source. Riehle et al. (2009) performed a survey with SAP developers to validate that IS helped them ot overcome intra-organizational boundaries. However, most of the surveyed literature neither validated the observed IS benefits nor discussed their generalizability. Thus, the presented IS benefits should be treated as *claimed* benefits and not as generally valid truth. We suggest further research to validate the reported IS benefits, evaluate their generalizability, and estimate the extent to which they affect the organizations adopting IS.

Table A.1 gives an overview of the seven identified benefits. Six of these benefits aggregate more fine-grained benefits.

Table A.1: Seven benefits of inner source



More efficient and effective development • Faster time-to-market • Reduced development costs	**Overcoming organizational unit boundaries** • Cost & risk sharing among org. units • Collaboration across org. unit boundaries • Program-wide information exchange
More successful reuse • Use of competences missing at component providers • Independence between reusers and providers • Relief of component providers	**Better Software Product** • Increased code quality • More innovative development
More flexible utilization of developers • Simplified developer deployment • Collaboration of detached developers	**Enhanced knowledge management** • Community-based learning • Openness and availability of knowledge
Higher employee motivation	

A.1 More Efficient and Effective Development

IS can result in a more efficient and effective development by reducing time to market, development cost, and generally increasing development efficiency (Riehle et al. 2016).

A.1.1 Faster time-to-market

Inner source enabled organizations to achieve a faster time to market. (Dinkelacker et al. 2002; Riehle et al. 2016) describes "faster development schedules with code leveraged among several products" as a benefit of IS. At Philips, it "led to time-to-market reduction of at least 3 months" (van der Linden 2009). Also, DTE Energy experiences quicker time-to-market (Smith and Garber-Brown 2007). Decreased time-to-market is a result of outside resources becoming available to component providers (Riehle et al. 2009) and shifting time lines by the possibility of using existing code and features earlier (van der Linden 2013).

Van der Linden (2013) attribute the faster-time-to market to the possibility to make earlier use of software that is not internally released yet: "Departments can already start developing upon new features of the platform before it is completely tested. This improves the time to market drastically."

A.1.2 Reduced development cost

IS can reduce the cost for software development and maintenance. HP (Dinkelacker et al. 2002; Melian and Mähring 2008), Lucent (Gurbani et al. 2006), DTE Energy (Smith and Garber-Brown 2007), and Philips (Wesselius 2008) reported a decrease in development costs. HP (Melian and Mähring 2008) experienced and Neus and Scherf (2005) assume increased development efficiency.

A.2 Overcoming of Organizational Unit Boundaries

Boundaries between organizational units (org. units) can become hard to cross in large organizations. By creating an intra-organizational community, IS is a vehicle to overcome such boundaries and raise awareness of company-wide activities and goals (Martin and Aitken 2012). At SAP, a majority of respondents (55/83) of a survey initiated by the founders of their IS program reported that IS enabled them to gather an understanding of other org. units' work (Riehle et al. 2009). IS facilitates an improved organization-wide perspective and intra-organizational collaboration (Riehle et al. 2016).

A.2.1 Cost and risk sharing among org. units

IS strengthens an organization-wide focus by promoting cost and risk sharing between org. units. Wesselius (2008) describes his experience:

> "It's indeed the cost- and risk-sharing benefits that primarily drive our ISS community - and both benefits reflect our overall business goals."

A consequence can be increased trust between org. units.

Cost and risk sharing between org. units can be achieved with collaborative development projects where org. units jointly start IS projects. Each org. unit supplies a fraction of the resources needed for the project. Consequently, cost and risk are shared among the org. units.

A.2.2 Collaboration across org. unit boundaries

IS enables collaboration among the org. unit boundaries to a degree not possible in traditional setups (Vitharana et al. 2010; van der Linden 2013). Collaborations among org. units' boundaries are more flexible in IS, enabling to quickly start, stop, and change collaboration (van der Linden et al. 2009).

In IS programs with free choice of components or even free task choice, a developer can quickly switch which ISS component to use or contribute to. In IS programs with assigned tasks and components, reusing parties still have the chance to decide how intensely and to which functional areas to contribute.

A.2.3 Program-Wide information exchange

Easy access to information spread over the organization is one of the main consequences of IS adoption (van der Linden et al. 2009). Vitharana et al. (2010) of IBM report that "findings reveal that the greater openness [...] enhances information sharing among a project's stakeholders". IS lowers the transaction cost for information (Neus and Scherf 2005). Eased information exchange leads to an increased awareness of organization-wide development efforts (Lindman et al. 2008).

A.3 More Successful Reuse

The surveyed literature reported IS's openness to enhance firm internal software reuse. DTE Energy observed IS to be a superior approach to embedding software reuse compared to purely tool driven strategies (Smith and Garber-Brown 2007; Anthes 2005). The awareness of other developer's activities (Vitharana et al. 2010) and the availability to pick up code on different levels of granularity (Whittaker et al. 2012) distinguish IS from other approaches to software reuse.

With a growing IS portfolio, more code becomes openly available to be picked up for reuse. Consequently, we believe selective or even universal IS programs to be more beneficial for enabling organization-wide software reuse than project-specific IS programs.

A.3.1 Use of competence missing at component providers

IS allows component providers to utilize competences and resources outside their organizational scope (Gurbani et al. 2006, 2010) and enables bottom-up collective intelligence (Riehle et al. 2009). This translates can translate into higher quality components and enables reusers to make ISS components fit better for their use-cases by contributing to them.

A.3.2 Independence between reusers and providers

In a traditional development setup, reusing a software component increases the dependence on its providers. IS decreases the dependence of reusers on providers. Reusers have the option to perform changes on their own in case the component providers have different plans regarding the components future (Vitharana et al. 2010; van der Linden 2013).

It is even possible to fork a project or perform and maintain one's own local modifications (Stol et al. 2011). As a result, political power play is mitigated (Gurbani et al. 2006). However, forking should be only done as a last resort because maintaining multiple forks of the same ISS component is costly and jeopardizes efficiency benefits of IS (Gurbani et al. 2006, 2010).

A.3.3 Relief of component providers

Component providers can become a bottleneck (Oor et al. 2008). IS allows the reusing parties to submit their own changes without having to wait for the component providers to implement them. Providing and maintaining components becomes less resource intensive (Vitharana et al. 2010).

Van der Linden (2013) sees this benefit as one driving force behind the IS adoption at Philips:

> "The most important reason for Philips to move to inner source development was to resolve the organisational issue that domain engineering was becoming a bottleneck in product line development. Increasingly more business units are using the platform developed by the domain engineering group."

Van der Linden (2013) summarizes that "inner source helped to break the platform bottleneck, since using departments are able to create patches".

A.4 Better Software Product

Organizations reported that IS enabled them to achieve a higher quality software product than with traditional development methods alone.

A.4.1 Increased code quality

IS is believed to result in increased code quality (Goldman and Gabriel 2005; Smith and Garber-Brown 2007; Martin and Aitken 2012) for example, shown by a lower defect ratio (van der Linden 2013). Fox (2007) of IBM concludes that "sharing code tends to increase its robustness".

The increased code quality can be explained by Linus' law which states that "given enough eyeballs, all bugs are shallow" (Raymond 1999). Linus' law has also validity in the context of IS (Neus and Scherf 2005; Melian and Mähring 2008) as developers from the community take part in debugging tasks (Dinkelacker et al. 2002; Riehle et al. 2009).

Other causes can be observed. Dinkelacker et al. (2002) observed "improved quality levels of shared software as authors' reputations are at stake". Also, Riehle et al. (2009) found the quasi-public scrutiny to make developers "feel compelled to strive for high quality of contributions". Finally, the increased employee motivation (see paragraph *increased employee motivation*) can result in higher code quality (Riehle et al. 2009).

A.4.2 More innovative development

The surveyed case studies reported that IS can lead to a more innovative development. Melian et al. (2002) attribute this to enhanced reuse coming with IS:

> "Tentative results indicate that [inner source] and its precursors facilitate collaborative efforts leading to improved conditions for software development, re-use and innovation within Hewlett-Packard."

IS enables firm internal open innovation (Morgan et al. 2011) and according to Riehle et al. (2009) enhances research-to-product transfer:

> "Research projects can get expertise and volunteers from downstream product units. Such early buy-in from the product units eases the research-to-product-unit transfer."

A contributor might even directly add innovative features to a component (Riehle et al. 2009).

Exploration-oriented IS projects like the ones at Lucent (Gurbani et al. 2006) or SAP (Riehle et al. 2009) can be used to explore innovative fields and consequently to enhance the research-to-product transfer. We believe a high-degree of self-organization (free choice of tasks and components) to support a more innovative development as developers are given opportunities to contribute new ideas and features to IS projects outside the immediate scope of their everyday work.

A.5 More Flexible Utilization of Developers

Riehle et al. (2016) observe developers in an IS context can be used more flexibly leading to improved resource management.

A.5.1 Simplified developer deployment

IS makes project information openly available and consequently can ease the deployment of individuals to new projects (Melian et al. 2002). Developers "can quickly join a project by understanding the rationale behind some feature selection and implementation" (Melian et al. 2002). In this way IS "creates an opportunity for rapid re-deployment of developers not just from one project to another but from one product to another" (Dinkelacker et al. 2002). Also the unified tool set often found in IS, eases deployment of developers (Dinkelacker et al. 2002; Riehle et al. 2009; Whittaker et al. 2012).

The ease to deploy developers between projects and tasks depends on the prevalence of the IS program in the organization. In a selective or universal IS program, developers switching projects benefit from the openly available project information in many IS projects. In a project-specific IS program, only one IS project exists. The ease of deployment between the potentially many non-IS projects does not necessarily change as a result of IS.

A.5.2 Collaboration of detached developers

IS allows detached developers to collaborate. Due to its open communication mechanisms, developers can collaborate even though they are geographically (van der Linden et al. 2009; van der Linden 2013) or temporally (Melian and Mähring 2008) detached.

A.6 Enhanced Knowledge Management

IS can lead to a better knowledge management as it allows knowledge dissemination by community-based learning and increases availability of knowledge. IS leads to enhanced intra-organizational knowledge sharing (Riehle et al. 2016).

Selective or universal IS programs make multiple IS projects and their documentation openly available. Consequently, such IS programs also make available knowledge regarding multiple IS projects. However, project-specific IS programs can also enhance the knowledge management within the organization. For example, the project-specific IS program around Lucent's SIP server implementation reportedly helped to convey knowledge about the back-then new SIP protocol to the organization's developers (Gurbani et al. 2006, 2010).

A.6.1 Community-Based learning

IS enables community-based learning within program-wide and project-specific communities. At Philips, one formal help desk group was completely replaced by mailing lists and discussion groups (van der Linden 2013). IS spreads knowledge regarding ISS components through the organization by enabling developers to gain hands-on experiences with new technologies (Gurbani et al. 2006). Mailing lists, wikis, and forums can support community based learning (Smith and Garber-Brown 2007; Martin and Hoffman 2007).

A.6.2 Openness and availability of knowledge

Openness of code allows developers to learn from more experienced colleagues' code (Whittaker et al. 2012). In addition, open communication transforms communication contents into accessible artifacts of knowledge. Typical communication tools that enable such a persistence are forums or mailing lists (Martin and Hoffman 2007). The persistent communication as well as open documentation artifacts in IS can result in constantly up-to-date documentation (Melian and Mähring 2008) and subsequently enhanced openness and availability of knowledge.

A.7 Higher Employee Motivation

IS can facilitate higher motivation and job satisfaction of developers (Riehle et al. 2016) and lead to "improved [...] morale and retention" (Martin and Aitken 2012). The increased motivation can result in the volunteering where developers are intrinsically motivated to contribute to an IS project in their spare time – similar to some developers in open source (Gurbani et al. 2006).

Google published an experience report of a developer who reported to be motivated by the self-selection of tasks at Google (Google-Blog 2006). We believe that a high degree of self-organization (free task choice and free component choice) contributes to higher employee motivation.

Riehle et al. (2009) reported of volunteering in an exploration-oriented IS project. Gurbani et al. (2006) reported of volunteering in a IS project that started as an exploration-oriented IS project. We believe that exploration-oriented IS project might be particularly motivating for those developers that are interested in learning about new technologies or being part of projects they perceive innovative.

B

Additional Materials regarding the Patch-Flow Crawler

This section provides additional materials regarding the patch-flow crawler presented in chapter 4. We give source code listings of particularly important interfaces and provide a mapping of key concepts of the patch-flow crawler's domain model to the context of the software tools it communicates with.

B.1 Source Code Listings

This section gives code listings of important interfaces discussed in chapter 4.

B.1.1 ScmAdapter Interface

```
public interface ScmAdapter {

        /**
         * Fetches the CodeContributions in the given range from a previously specified
         * repository.
         *
         * The method can be run in parallel multiple times. But the caller needs to
         * ensure that it is only called for onw repository (as identified by
         * getId()) once. Otherwise, the state of the local cache will be corrupted
         * leading to wrong results.
         *
         * For simplicity, ScmAdapter allows its implementors to ignore the very
         * first commit done to a repository.
         *
         * @param _repository
         *            The Repository that we want to fetch the patches from
         *
         * @param _start
         *            The point in time the date range you are interested in begins
         *            (inclusive)
         * @param _end
         *            The point in time the date range you are interested in ends
         *            (exclusive; the first point to discard)
         *
         * @return Code contributions contributed in the defined time frame. For each,
         *         the following attributes are set:
         *
         *         1. Repository object (identifying the repository the contribution was
         *         made to)
         *
         *         2. The FileChange objects (regarding each code contribution)
         *
         *         The CodeContribution object returned by this method does not have
         *         other objects linked to it (like a patch's author, committer,
         *         inner source project etc.)
         *
         *         Some source code management systems already give you information
         *         on the persons authoring or committing changes. In this cases,
         *         implementors of this method are allowed to link Person objects as
         *         authors and committers but are responsible themselves to provide
         *         CodeContributionProcessors that persist these objects.
         */
        public Iterator<CodeContribution> fetch(Repository _repository, Date _start, Date _end);
}
```

B.1.2 PreStep and PostStep Interface

```
public interface PreStep extends Activity { /* no code */ }

public interface PostStep extends Activity { /* no code */ }

public interface Activity {

        /**
```

```
         * Run the defined step.
         *
         * @param _crawlRun
         *            CrawlRun the step belongs to.
         */
        public void run(CrawlRun _crawlRun) throws StepExecutionFailure;


}
```

B.1.3 PreStep and PostStep Interface

```
public interface CodeContributionProcessor {
        /**
         * Performs the specified processing on the given code contribution.
         *
         * @param _cc
         *            The code contribution to be processed (modified).
         * @return Process (modified) contrib. or null if the processor decides to
         *         filter out the CodeContribution.
         */
        public CodeContribution process(CodeContribution _codeContribution);
}
```

B.2 Mapping of Concepts

Table B.1 describes to what terms the concepts of the patch-flow crawler (and its domain model) map in the context of GitHub Enterprise, Gitlab, and Microsoft Team Foundation Server (TFS).

Table B.1: Mapping of concepts to terminology of GitHub Enterprise, Gitlab, TFS

Concept	**Evuivalent**		
	GitHub Enterprise	Gitlab	TFS
Code contribution	Commit	Commit	Change set
Patch	Commit (whose author is not part of org. unit owning the receiving IS project)	Commit (whose author is not part of org. unit owning the receiving IS project)	Change set (whose author is not part of org. unit owning the receiving IS project)
Repository	Repository	Repository	Repository
IS Project	Assumption: IS projects = repository	Assumption: IS projects = repository	N/A
Committer (of a code contribution)	A person identified by the committer pseudonym	A person identified by the committer pseudonym	N/A
Author (of a code contribution)	A person identified by the author pseudonym	A person identified by the author pseudonym	Author

C

Research Protocol for Multiple-Case Case Study

Chapter 5 presented a case study of three organizations running five inner source (IS) programs that identified their IS practices, measured the patch-flow, and theorized about the relationship of IS practices and patch-flow.

This appendix presents parts of our case study protocol. We shortened the case study protocol presented in this appendix to avoid duplication with those parts of the approach already discussed in chapter 5. Following the suggestions of Yin (2013) and inspired by other researchers' protocols (Stol and Fitzgerald 2014), we report on ...

- ... the study's objectives and our design decisions (section C.1)
- ... our data collection procedures and questions (section C.2)
- ... our guidelines for reporting the case study data and findings (section C.3)

In addition, this protocol summarizes key artifacts including ...

- ... our code book developed as part of the thematic analysis of IS practices (section C.4.1)

- … quotes from interviews in their original language (section C.4.2)

A case study protocol is a living document that is continuously adapted and refined during the course of the study. A sizeable portion of this protocol as it is now was written in late stages of our research.

This case study protocol is written with two audiences in mind: First, we wrote the protocol for ourselves (the researchers undertaking the case study). It serves as a guideline for us collecting, analyzing, and reporting on the case study data. It makes explicit how we set out to operate. Where things were left implicit in the beginning of our study, it forces us to retrospectively document our approach in more detail than in an article's method section – helping us to identify potential flaws in our study's trustworthiness. Second, we wrote the protocol for fellow researchers reading the corresponding article. We intent this protocol to support replication of our study and allow readers to judge its trustworthiness and limitations on a informed basis. As a consequence, the majority of this protocol is written in past tense.

C.1 Case Study Overview & Design

This section gives an overview of our case study and its underlying research design.

C.1.1 Research Objective

Our case study addresses the following research questions:

- RQ4: What is the magnitude of IS collaboration in organizations?
- RQ5: How do IS practices affect IS collaboration?

We discussed the research objectives, our motivation, an resulting research questions in detail in chapter 5. We omit further discussion of our research objectives in this appendix.

C.1.2 Case Study Research and Alternatives

Yin (2013) defines case study research as follows:

> "A case study is an empirical inquiry that investigates a contemporary phenomenon […] in depth and within its real-world context, especially when the boundaries between phenomenon and context my not be clearly evident."

We deemed case study research to be an appropriate method choice for our research question:

- *Boundaries between phenomenon and context unclear.* Case studies empirically investigate a phenomenon within its real world context (Yin 2013). This makes them particularly useful if the boundaries between a phenomenon and its context cannot be clearly defined. This is the case with the organizations we study: It was not immediatly clear if IS practices or also other contextual (e.g. the strong dependency of product units on the platform unit in medical org.'s imaging platform) influence IS collaboration or its magnitude.

- *Contemporary nature of phenomenon.* Case studies are particularly fit to investigate contemporary phenomenon. This is the case with the phenomena we study: We identified IS practices and IS collaboration happening at time of data collection or in the very near past (within one year of data collection).

- *No example answers of research questions.* No prior research has yet answered RQ4 and RQ5. Case study research is fit to systematically produce example: Particularly for RQ4, our case study creates examples by demonstrating what magnitude and structure of IS collaboration is possible and to be expected.

We identified no alternative feasible method:

One alternative to case studies are experimental research setups (controlled experiments, field experiments). We excluded experimental setups for the following reasons:

- Absence of theories to validate. Experimental setups are typically used to validate theories by means of hypothesis testing. With regards to our research questions, no preexisting theories or hypothesis were known to us (controlled experiments, field experiments).

- Labroratory setting unrealistic. Due to the amount of individuals involved in IS collaboration and their rich working context, we deemed it impossible to replicate IS like collaboration in a labroratory setting (controlled experiments).

- Higher degree of control unrealistic. We did not believe it to be realistic that organizations would allow us to manipulate the used IS practices according to our research agenda (field experiments).

Another alternative are multiple variants of survey research (source?) where a statistically diverse or even representative sample of a population is surveyed. In our situation, one could imagine to question a representative sample of all organizations using IS to identify what IS practices they use and how their patch-flow looks like. This was not feasible in the scope of this thesis for at least one reason: Patch-flow measurement is a non-trivial and costly (time consuming) undertaking. We did not expect survey participants to do this on their own; we did not have the resources to do it ourselves for tens (diverse) or hundreds (representative) organizations.

C.1.3 Case Study Design

We chose to perform case study research. Case study research is suitable to empirically investigate a contemporary phenomenon in its real-life context (Yin 2003). Case studies can be used when the boundaries between the phenomenon and context are not clearly evident (Yin 2003). Different classes of case studies exist.

Multiple-Case CaseStudy

Case studies can be designed as single-case or multiple-case case studies (Yin 2013): In a multiple-case design, "two or more cases within different contexts make up a multiple[-case] case study" (Runeson et al. 2012). In a single-case design, only one case exists.

We studied three organizations which largely independent contexts. While two of the selected organizations are in an association relationship (one organization holds owns parts of the other), all three organizations are independently managed, report their results independently of one another, and are each listed on a stock market separately. With regards to our research objective, we consider the contexts of all three organizations to be independent and constitute separate cases. We consider our case study a multiple-case case study.

Embedded Case Study

In a holistic design, one unit of analysis per case is considered. The case itself is the unit of analysis. In an embedded design, multiple units of analysis per case are considered (Yin 2013). A unit of analysis is "unit lesser than the main unit of analysis [the case], from which case study data also are collected" (Yin 2013). Runeson et al. (2012) summarize that "the holistic design is more appropriate when there are no logical subunits to the case, and therefore no obvious additional units of analysis". Units of analysis are different from an isolated case in that two units of analysis can share a common context while to cases have an isolated context.

We consider our case study an embedded case study. Two of our three case organizations have more than one independent IS program. While these programs share the same context (the same organization), they are organized differently and IS happens using differing practices and to differing extent. This warrants to analyze each IS program as individual unit of analysis. We initially assumed our case study would follow a hollistic design investigating IS practices on a per case (per organization)but learned about the different IS programs during our the course of our data collection.

What one considers an isolated context or a unit of analysis depends on the research objective or question at hand (Yin 2013).

Exploratory Case-Study

According to Runeson et al. (2012), case studies can serve multiple purposes. A case study can be ...

- *Descriptive* - "portraying the current status of a situation or phenomenon"
- *Exploratory* - "finding out what is happening, seeking new insights, and generating ideas and hypothesis for new research"
- *Explanatory* - "seeking an explanation for a situation or a problem, mostly but not necessarily, in the form of a causal relationship"
- *Improving* - "trying to improve a certain aspect of the studied phenomenon"

We position our case study as an exploratory case study because we seek to generate new insights (in the form of hypotheses about the relationship between IS practices and IS collaboration). However, our study also has descriptive and explanatory aspects as outlined below.

Descriptive. When answering RQ4, our case study is descriptive in that it does not form a theory on the magnitude of IS collaboration (using patch-flow as a proxy) but is simply describing our observations and thus portreying the state of the patch-flow phenomenon in the cases' units of analysis.

Explanatory. When answering RQ5, our case study is explanatory in that we seek to identify relationships between two phenomena (IS practices exercised and IS collaboration observed). However, it is not our primary goal to explain the relationships we observed.

Exploratory. Rather than explaining, we uncover correlations and interpret as well as hypothesize about their meaning and reasons. We present hypotheses forming a theory of the effect on IS practices on IS collaboration. Thus, we used our case study for theory building (Eisenhardt 1989) and it serves an exploratory purpose.

Case Selection

We discuss the case selection in detail in chapter 5. We omit further discussion in this appendix.

C.2 Data Collection Procedures & Questions

This section describes our data collection procedures.

C.2.1 Qualitative Data

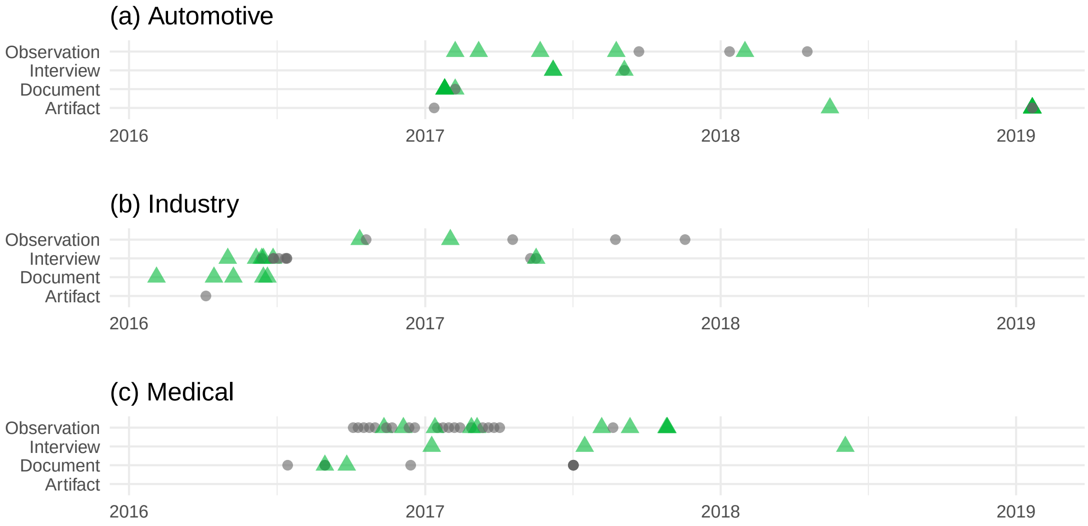
To understand how IS programs are set up in the studied organizations, we collected qualitative data. Subsequently, we performed a thematic analysis (Braun and Clarke 2006) of the data to identify collaboration practices. For each identified collaboration practice, we assessed whether it is an IS practice or not using the findings of prior literature.

In this section, we outline the time frame over which data was collected, how we establish informed consent, and give an example of a used interview guideline. We will omit a general description of the data collection and analysis proceedure because it is layed out in chapter 5.

Collection & Inclusion

Figure C.1 shows all collected qualitative data items. The x-axis shows the date on which we collected the item. The y-axis indicates the data collection approach. Each green triangle represents an item considered in our case study. The figure shows that we started data collection in the case organization with timely distance: The industry organization served as our pilot study. Based on our learnings, we adjusted the data collection procedures in the automotive

Figure C.1: Overview of collected and included qualitative data



and medical organization (i.e. by reducing the number of interviews and increasing the number of opportunities for direct observations).

Figure C.1 also shows as gray circles excluded items not considered in the case study. We manually read and inspected every item to decide on inclusion or exclusion. We excluded observation notes (three in the automotive, four in industry, 20 in the medical organization) because nothing of relevance to the case study was discussed during the observed meetings and workshops. We also excluded one interview in the automotive organization because the interviewee (suggested by the organization) was a new hire who did not yet have an understanding of the IS program. All other excluded items, were excluded because they had largely overlapping content with other items or saturation was already reached.

The figure demonstrates our prolonged engagement (Guba 1981) with the case organizations. It also documents a predicament: Much of the observations we performed, particularly at medical org., were not deemed not useful for the study. This is, because we observed meetings and workshops where nothing of interest to this study was discussed. Still, we deemed it worthwhile to participate: It allowed us to forge a trust relationship to the organization, we collected data that might be useful for future research, and as researchers understood the context of the organization and their culture better.

Informed Consent

To establish informed consent with interviewees (Singer and Vinson 2002), we informed them about the interview and asked their permission to perform an audio recording at least 24 hours before the interview. We used the following email template:

> Sehr geehrte Frau XXXXXXX,
>
> Wir freuen uns schon auf das gemeinsame Gespräch mit Ihnen am XXXXX dem XX.XX. um XX:XX.
>
> Wir sind Forscher in der Open-Source-Research-Group an der Friedrich-Alexander-Universität Erlangen-Nürnberg. Auf Initiative von XXXXX XXXXXXXXX möchten wir erforschen wie und ob die Zusammenarbeit am XXXXXXXXXXXXXXX mit Inner Source verbessert werden kann.
>
> [...]
>
> Für eine Bestandsaufnahme möchten wir bei unserem gemeinsamen Gespräch mehr aus Ihrem Arbeitsalltag und zur Zusammenarbeit mit anderen Teams erfahren. Dazu haben wir einige Fragen vorbereitet. Hierzu gibt es keine falschen oder richtigen Antworten. Vielmehr interessiert uns Ihre Sichtweise und Ihre Erfahrungen.
>
> Für eine präzise wissenschaftliche Auswertung der Inhalte, möchten wir gern eine Audioaufzeichnung des Gespräches vornehmen. Audioaufzeichnungen und Gesprächsprotokolle werden von uns selbverständlich vertraulich behandelt und auf speziellen Servern archiviert.
>
> Falls von Ihnen gewünscht, können wir auf eine Audioaufzeichnung verzichten und das Gespräch händisch protokollieren. Dieses Verfahren ist allerdings fehleranfälliger und weniger detailliert; es könnte die Qualität unserer Ergebnisse mindern.
>
> Falls von Ihrer Seite vorab Fragen bestehen, zögern Sie bitte nicht uns zu kontaktieren.
>
> Mit freundlichen Grüßen, XXXXXXXX XXXXXXXX

The following is an English translation of this template:

> Dear Mrs. XXXXXXX,
>
> We are already looking forward to our conversation with you at XXXXXX XXXX.XX. at XX.XXX.
>
> We are researchers from the Open-Source-Research-Group of the Friedrich-Alexander-University Erlangen-Nürnberg. As part of an initiative by XXXXX XXXXXXXXX we would like to investigate whether and how the collaboration on XXXXXXXXXXXXXXX with Inner Source can be improved.
>
> [...]

> For an assessment we would like to more about your everyday work and the collaboration with your colleagues during our conversation. For this, we prepared some questions. There are not wrong or right answers. Much more, we are interested to learn your perspective and experience.
>
> For a precise scientific analysis of the contents, we would like to perform an audio recording on the conversation. Audio recordings and protocols will of course be treated confidential and archived on specific servers.
>
> If you prefer, we can refrain from doing an audio recording and manually write a protocol of the conversation. However, this approch is more error prone and less detailed; it could limit the quality of our results.
>
> If you have any questions, please do not hesitate to contact us.
>
> Best regards, XXXXXXXX XXXXXXXX

Interview Scheduling

We omit our interview schedule for privacy reasons.

C.2.2 Quantitative data

Chapter 5 gave an overview of how we implemented the steps of the patch-flow method to measure patch-flow. Table C.1 gives a more detailed overview of the steps performed in each organization. The numbers in the first column correspond to the steps present in chapter 3.

C.3 Guide for Case Study Report

We intend to report on the case study with two different publications:

1. A chapter of a thesis (this thesis) reporting about the case study findings
2. A journal-length article

In addition to describing typical items like related work or our research approach, we report on the following insights generated from the case studies:

- We deliver a thick description of our observations in the case study. Per unit of analysis we report on the observed...
  - ... the collaboration practices (including IS practices and non-IS practices)

Table C.1: Patch-flow measurement steps per organization

	Automotive Org.	**Industry Org.**	**Medical Org.**
(1) Extraction of code contributions	We extracted code contributions from the organization's GitHub Enterprise forge API using our patch-flow crawler.	An engineer of the organization utilized a proprietary export script to extract code contributions from the organization's Team Foundation Server (TFS) instance.	For the development tools, we extracted code contributions from the organization's TFS instance's API using our patch-flow crawler. For the imaging platform, an engineer of the organization utilized proprietary export scripts to extract code contributions from the TFS instance.
(2) Mapping of code contributions to IS projects	In GitHub Enterprise, each IS project is stored in its own repository making the mapping of code contributions to IS projects trivial.	We utilized the directory paths to determine for each code contribution the receiving IS project.	We utilized the directory paths to determine for each code contribution the receiving IS project.
(3,4,5) Identification of authors and mapping to org. units	For a majority of code contributions, either a unique ID of the author or his/her email address was available. We queried an LDAP server storing information on all employees using the unique ID or the email address to identify authors and their org. unit.	Each TFS export contained a unique identifier of the author of a code contribution. Due to privacy concerns, we did not get access to the employee database but an engineering manager at the organization manually looked up these unique identifiers in a firm-internal database, assembled information on the organizational units into a machine readable format, and mapped authors to the organizational units.	For a majority of code contributions, either a unique ID of the author or his first name, last name, and department code were available. The company provided to us a list in XML format with information on select developers. We identified authors and mapped them to organizational units using this list, the information available with the code contributions.
(6) Mapping of IS projects to org. units	We mapped each IS project to the org. unit whose developers contributed the most code contributions to it.	We mapped each IS project to the org. unit whose developers contributed the most code contributions to it.	We mapped each IS project to the org. unit whose developers contributed the most code contributions to it.

  - … the IS collaboration (using patch-flow as a proxy)

- We perform a cross-unit of analysis synthesis where we compare the IS practices and patch-flow found in each unit of analysis.

- We discuss our interpretation of the observations separately from the case study observations.

When reporting the results in a length-constrained outlet, we drop the thick description of the unit of analysis and emphasize the (less space consuming) synthesis. This is consistent with the reporting format for multiple-case case studies suggested by Yin (2013):

> "Your full multiple-case report will consist of the single cases, usually presented as separate chapters or section. In addition to these individual cases, your full report will contain an additional chapter or section covering the cross-case analysis and results. As another common variant, the cross-case material can form the bulk of the main report (especially suitable for a journal-length article) [...]"

Where possible we give empirical evidence by quoting from the qualitative data items or at least referencing which qualitative data items we are refering to.

C.4 Artifacts

This section presents key artifacts created or collected as part of the case study. We present our code book (used as part of the thematic analysis to identify IS practices).

C.4.1 Code Book

As described in chapter 5, we analyzed the collected qualitative data using thematic analysis (Braun and Clarke 2006). To support this process, we created a code book (Guest et al. 2006). The following table C.2 presents our code book.

Table C.2: Codebook used for thematic analysis

Code	Full definition	When (not) to use
Theme: 1) Open code for all parties to read, (re)use		
All parties can read code	Every party in the organization can see the source code of an IS project.	When there is evidence that every party in an organization can see the code of an IS project or when an interviewee does mention seeing the code but does not mention limitations.
All parties can read documentation	There are documents that every party in the organization can access	When there is evidence that every party in an organization can read documentation artifacts regarding an IS project.
All parties can (re)use	Every party in the organization can reuse the code of an IS project	When there is evidence that every party in an organization can reuse the code of an IS project or when an interviewee does mention reuse but does not mention limitations.
Theme: 2) Provide single entry point to IS program		
Feature for searching IS projects	The software used to run / host the IS program offers features for searching IS projects.	Use when there is indication that there is a software that offers features to perform a search (i.e. keyword search) to find IS projects. This code can also be used of no dedicated forge software is used.
Feature for listing IS projects	The used software forge offers features for listing IS projects.	Use when there is a list of all or specifically filtered IS projects. The code can also be used if the list is not created automatically but e.g. curated by a responsible individual or group.
Theme: 3) Open code to read, (re)use only for selected parties		
All parties can read code	See above!	

Table C.2: Codebook used for thematic analysis - continued

Code	Full definition	When (not) to use
Product line participants can (re)use	Only participants in a specific product line can (re)use code of the IS program	Use when there is indication that only product line participants can participate in (re)use. Do not use for purely technical reasons parties are excluded from (re)using.
Theme: 4) Open code for contributions by all parties		
All parties can contribute patches	Every party in the organization can contribute code to an IS project	When there is evidence that every party in an organization can contribute code to an IS project or when an interviewee does mention code contributions but does not mention limitations.
Theme: 5) Establish committer role		
Committer role enforced by forge	No committer role is explicitly defined, but the feature of the forge force users to define a committer (or multiple committers) per project	Do not use when there is indication of a committer role formally existing (e.g. in process documents, descriptions of how to run an IS project).
Mandatory review by committers	All code contributions (by a person who is not the committer of the project) have to be reviewed by a committer	Use when there is indication that code review is mandatory by a committer and the role of committer is explicitly defined per IS project.
Mentoring by committers	Committers mentor contributors	Use when there is evidence that committers are not only refereeing patches but are also providing (implicit or explicit) mentoring to individuals e.g. by giving feedback on a patch submission.

Table C.2: Codebook used for thematic analysis - continued

Code	Full definition	When (not) to use
Theme: 6) Perform ad hoc code review & mentoring		
Ad hoc review by arbitary developers	Code contributions are reviewed in an adhoc fashion and not by designated persons (i.e. not by designated committers)	Use when there is evidence that code review happens in an adhoc fashion (sometimes they do, sometimes they don't) or by arbitrary / contributor-selected people. Do not use when there is a defined process for review (e.g. we always perform pre commit review by persons with defined properties) or defined role (e.g. a committer reviews everything).
Review not mandatory	Code review of contributions is not mandatory	Use when there is indication that code review is not mandatory. Do not use when interview / document refers to ad hoc or voluntary code review.
Ad hoc mentoring by arbitrary developers	Contributors are mentored in an adhoc fashion	Use when there is evidence that mentoring happens in an adhoc fashion (sometimes it does, sometimes it doesn't). Do not use when there is a defined process for mentoring (i.e. committer mentors newbies).
Theme: 7) Establish mandatory code review (but not committer role)		
Mandatory review by arbitrary developers	Code contributions are always reviwwed by not by designated persons (i.e. not by designated committers)	Use when there is indication of mandatory review (e.g. enforced by process or infrastructure) but there is no indication of a dedicated reviews (i.e. a committer).

Table C.2: Codebook used for thematic analysis - continued

Code	Full definition	When (not) to use
Theme: 8) Open code for contributions by selected parties		
One specific org. unit can contribute	Only specific org unit can contribute to the IS program	Use when there is evidence that only one specific org. unit can contribute code to an IS project. Do not use when there are technical reasons or temporal reasons for others being excluded. Do only use when this is an explicit decision (e.g. for process reasons).
Theme: 9) Provide dedicated open communication infrastructure		
Dedicated open communication infrastructure	There is dedicated infrastructure for open communication.	Use when there is evidence of infrastructure being provided for the sole or primary reason of enabling open communication.
Theme: 10) Communicate using closed communication		
Closed communication	Closed communication (communication that is not open communication) regarding IS programs or projects is excercised.	Use when there is evidence of non-open communication regarding IS projects or the IS program.
Incidental open communication	Incidental open communication is taking place regarding IS programs or projects.	Use when there is evidence of incidental / sparsely happening open communication.
Theme: 11) Establish governance committees (inspired by OS foundations)		
Project management committees	Project-specific project management committees are responsible for managing the individual projects.	Use when there is evidence of a committee / board as described in the definition.
Program-wide operational board	A program-wide operational board governs / runs everyday activities of the program.	Use when there is evidence of a committee / board as described in the definition.
Program-wide steering board	A program-wide steering committee is responsible for the strategic decisions in the program	Use when there is evidence of a committee / board as described in the definition.

Table C.2: Codebook used for thematic analysis - continued

Code	Full definition	When (not) to use
Theme: 12) Design & enforce IS license		
License for each IS component	Every IS project is using a license that is pre-defined and the same for all projects.	Use when there is evidence of an IS license a) existin and b) being mandatory for all projects in the IS program.
Contributor license agreement	Every contributor has to sign an individual contributor license agreement.	Use when there is indication that project contributors have to sign a contributor license agreement first.
Theme: 13) Select & coach IS projects using project incubator		
Project incubator	A project incubator selects projects, makes explicit the project lifecycle status, and helps projects to grow their communities	Use when the is evidence of a project incubator as described in the definition.
Existing code base	A code base must exist for a project to be included into the program	Use when there is evidence that a project must have an existing code base to be included into the IS program.
Multiple interested parties	For the project to be included, multiple parties must be interested (or potentially interested) in the project.	Use when there is evidence that a project must have multiple parties interested in it to be included into the IS program.
Theme: 14) Establish informal governance mechanisms		
Regular coordination meetings	Regular meetings regarding the IS program take place	Use when there is evidence of IS program governance by means of a regular meeting.

C.4.2 Translated Quotes

The interviews we performed as part of the case study were all performed in a language other than English. As a consequence, chapter 5 presents quotes from the interviews translated to English. The following table C.3 presents the original quote for each translated quote.

We redacted information that allows a reader to identify our case organizations or individuals within the cases. For example, when an individual mentioned a product or project name, we refer to this as "[specific project]". When an individual named a person, we refered to him or her as "[specific person]".

Table C.3: Original quotes and English translation

Item	Translation	Original
A_{I2}	"We designed [AutoSource] and every employee of [automotive org.] can participate. Its public and everybody can see what we do – everyone else within the company."	"Wir haben [AutoSource] aufgesetzt und ganz [Automotive org.] kann jetzt auch daran teilnehmen. Das ist öffentlich und jeder sieht auch, was wir machen sieht auch jeder andere innerhalb der gesamten Firma."
A_{I2}	"Well, allright, this is not only present in [AutoSource]. But [there is] collaboration where you have a shared repository, where everybody sees what others are doing there, and you self-organize"	"Ja gut das ist jetzt nicht ausschließlich [AutoSource] aber die Zusammenarbeit in der Form, dass man eben das gemeinsame Repository hat, jeder sieht, was die anderen darauf machen und man organisiert sich dann auch."
A_{I4}	"There are – but that is a technical problem – excluded parties [...]. They use another technical infrastructure than the remaining group."	"Es gibt - das ist aber technisches Problem - Parteien die ausgeschlossen sind, weil sie von der Plattform abgehalten werden aus technischen Gründen [...]. Die nutzen eine andere technische Infrastruktur als Automotive. "
I_{I2}	Referring to an IS project: "There is surely one or the other thing that we contributed because it was simply not yet available or because we have a solution already within our environment."	"Und es gab sicherlich das eine oder andere [Feature], das wir beigesteuert haben, weil es noch nicht zur Verfügung stand oder weil es in unserem Umfeld zuerst da war."
I_{I2}	"If we do our own work on a [specific IS project], then we do not just check in our contribution, but we write to [a specific employee], who is responsible for [the specific IS project] and ask him: 'We had to adapt something. Can you have a look at that?'"	"wenn wir eben selbst [spezifisches-Projekt]-Sachen anpassen müssen, dann checken wir die nicht einfach ein, sondern wir schreiben den [einem bestimmten Mitarbeiter], der auch die Verantwortung für die [dieses spezifische Projekt] hat, an, und sagen 'Wir haben hier etwas anpassen müssen. Kannst du dir das angucken?'"

Table C.3: Original quotes and English translation - continued

Item	Translation	Original
I_{I2}	"We at the [specific] team, we have the habit of at least asking those responsible for an [IS project] to perform a review."	"Wir vom [spezifischen] Team haben uns angewöhnt, die [Projekt-]Verantwortlichen zumindest mal zu einem Review anzuschreiben."
I_{I2}	"The [...] code we develop: It's for everyone. We just have the responsibility for it."	"Der [...] Code, den wir entwickeln. Der gehört der Allgemeinheit. Wir haben die Verantwortung."
I_{I3}	"Everybody can influence [a project] at any point in time: either reporting a bug or contributing an extension"	"Jeder kann jederzeit Einfluß darauf nehmen, Fehler melden oder Erweiterungen weitergeben."
I_{I3}	"It's not mandatory that I have to do a review for each code contribution. Not mandatory. No."	"Es ist nicht die Pflicht, dass ich wirklich unter jeder Code-Änderung einen Review machen muss. Die Pflicht ist nicht da, nein."
I_{I3}	"[There is] a [test infrastructure] planning meeting. What's going on? What's the status? How to continue regarding tooling? For example. So, there are different things put on the agenda."	"[Es gibt] so eine [Testinfrastruktur]-Planungsrunde: wie es aussieht? Der Status? wie es weitergeht bezüglich Tooling? Zum Beispiel. Also es kommen verschiedene Punkte auf die Tagesordnung drauf."
I_{I3}	"Those [projects] are libraries – ready made functionality – that everybody can use. Not only our [team], but everybody. [...] I have read access. Also other teams can look at my source code. That's what it [the program] is made for"	"Das [die Projekte] sind Bibliotheken - vorgefertigte Funktionen - die jeder einsetzen kann. Also nicht nur unsere [Teammitglieder], sondern jeder. [...] Leserechte habe ich. Auch andere Teams können in meine Sourcen reinschauen. Dafür ist es ja gedacht."
I_{I4}	"I have full access on the source code [of the test infrastructure projects]"	"Also auf die [Testinfrastruktur] Sachen habe ich vollen Zugriff auf die Sourcen"
I_{I4}	"In principle every body can check in [changes]. If it builds, it's in."	"Im Prinzip kann jeder einchecken. Und wenn es nachher baut, dann ist es drin."
I_{I4}	"They communicate among themselves, using communicator, email."	"Nein. Die kommunizieren unter sich, eben auch Communicator, E-Mail."
I_{I5}	"There are regular status meetings for developers [of test infrastructure components and tools]"	"Es gibt regelmäßige Statusrunden für [Testinfrastruktur-Komponenten und Tool]-Entwickler."

Table C.3: Original quotes and English translation - continued

Item	Translation	Original
I_{I5}	“There is nobody who examines the code, one checks in. You got to know what you are doing. And well, you check it in [into the repository] and then it’s in”	“Es gibt auch niemanden der den Code prüft, den man da eincheckt. Man muss wissen, was man tut. Und ja, das wird dann halt eingecheckt und dann ist es drin.”
I_{I5}	“Typically via communicator [an instant messaging tool], email, sometimes using face to face meetings. If somebody works in the same building, you get together.”	“Normalerweise über Communicator, E-Mails, manchmal auch direkt treffen. Wenn es jemand hier im Haus ist, dann setzt man sich direkt zusammen.”
I_{I5}	“We fix bugs in [component a], [component b], [component c] ourselves as well.”	“Wir fixen einige Bugs in [Komponente a], [Komponente b], [Komponente c] auch selber. ”
M_{I1}	“If a colleague doesn’t react to a review ticket within two hours, the author will typically search for another reviewer”	“Wenn ein Kollege sich binnen 2h nicht zu einem Review Ticket meldet, sucht der Autor für gewöhnlich einen anderen Reviewer” (from hand written notes)
M_{I1}	“Reviews of various artifacts are mandatory. [...] Product code must be reviewed by at least on additional software developer”	“Reviews von verschiedenen Artefakten verpflichtend. [...] Product Code (mindestens ein weiterer Entwickler)” (from hand written notes)
M_{I1}	“The developer can pick the reviewer himself [...] Often you simply ask somebody from your own team”	“Entwickler kann Reviewer selbst auswählen [...] Normalerweise Reviewer aus eigenem Team” (from hand written notes)
M_{I3}	“If I am a developer of [a product unit], and I want to contribute to the [platform], I can’t just readily do it. [...] We designed a special delta-training with which a [product-]developer receives the necessary privileges – from the process perspective – by means of an additional training to change things at the [platform].”	“Wenn ich jetzt ein Entwickler von [Produkteinheit] bin, und ich möchte auch in der [Platform] etwas dazu beitragen, dann kann ich das nicht so ohne weiteres tun [...] [Wir haben] eine spezielle Delta-Schulung sozusagen ins Leben gerufen, mit der ein [Produkt]-Entwickler durch eine zusätzliche Schulung - aus Prozess-Sicht - dazu berechtigt ist, in der [Platform] etwas zu ändern.”

Table C.3: Original quotes and English translation - continued

Item	Translation	Original
M_{I_3}	"So, mandated by our software process we have the obligation to review. Both handing in the code for review and performing the code review [...] has to follow specific guidelines."	"Also wir haben aus unserem Software-Prozess heraus eine Verpflichtung etwas zu reviewen. Sowohl das Code -Abgegeben, als auch das Reviewen des Codes [...] muss nach gewissen Richtlinien erfolgen."