Inner Source Definition, Benefits, and Challenges

MAXIMILIAN CAPRARO and DIRK RIEHLE, Friedrich-Alexander-Universität Erlangen-Nürnberg

Inner Source (IS) is the use of open source software development practices and the establishment of an open source-like culture within organizations. The organization may still develop proprietary software but internally opens up its development. A steady stream of scientific literature and practitioner reports indicates the interest in this research area. However, the research area lacks a systematic assessment of known research work: No model exists that defines IS thoroughly. Various case studies provide insights into IS programs in the context of specific organizations but only few publications apply a broader perspective. To resolve this, we performed an extensive literature survey and analyzed 43 IS related publications plus additional background literature. Using qualitative data analysis methods, we developed a model of the elements that constitute IS. We present a classification framework for IS programs and projects and apply it to lay out a map of known IS endeavors. Further, we present qualitative models summarizing the benefits and challenges of IS adoption. The survey provides the first broad review of IS literature and systematic arrangement of IS research results.

Categories and Subject Descriptors: A.1 [**Introductory and Survey**]; D.2.9 [**Software Engineering**]: Management—*Lifecycle; Productivity; Programming teams; Software process models*; D.2.13 [**Software Engineering**]: Reusable Software—*Reuse models*; K.6.1 [**Management of Computing and Information Systems**]: Project and People Management

General Terms: Economics, Human Factors, Management

Additional Key Words and Phrases: Inner source, taxonomy, open collaboration, internal open source, software development methods, software engineering, software development efficiency, software development productivity

ACM Reference Format:
Maximilian Capraro and Dirk Riehle. 2016. Inner source definition, benefits, and challenges. ACM Comput. Surv. 49, 4, Article 67 (December 2016), 36 pages.
DOI: http://dx.doi.org/10.1145/2856821

1. INTRODUCTION

Open source software plays a crucial role in today's software industry. Open source development tools help to build software and open source components are used as part of proprietary software. Organizations employ a variety of strategies and business models to get involved with or benefit from open source communities [Riehle 2007, 2009]. Open source is recognized to be capable of delivering high quality software [Crowston et al. 2008].

Consequently, researchers and software developing organizations desire to know how the success of open source is possible and how industry could benefit not only from its

This work was partially funded by a Black Duck Software Inc. grant to the Open Source Research Group at Friedrich-Alexander-Universität Erlangen-Nürnberg.
Authors' address: M. Capraro and D. Riehle, Friedrich-Alexander-Universität Erlangen-Nürnberg, Open Source Research Group, Martensstr. 3, 91058 Erlangen, Germany; emails: maximilian.capraro@fau.de, dirk.riehle@fau.de.

DOI: http://dx.doi.org/10.1145/2856821

outcomes (the software components and tools) but also from the development practices exercised in the open source world.

Inner source (IS), first mentioned by O'Reilly [2000], is an answer to this question. In IS, development practices and culture from the open source world are implemented within organizations. While the development is similar to and shares attributes with open source software development, the software remains proprietary and is only available to a defined group of developers (for example, the employees of an organization). A steady stream of scientific publications and other sources such as blog and magazine articles since 2002 indicates a vivid interest in this research topic.

The majority of scientific publications present case studies of IS in the context of one or a few organizations. 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 of IS programs, benefits, and challenges of IS adoption as well as practices to tackle these challenges were studied only in the context of selected organizations. The absence of models 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.

This article resolves the issue by assessing the state of the research and introducing a set of qualitative IS models that provide a unified view of IS research results. For this survey, we considered a total of 43 scientific publications plus additional materials. In detail, the contributions of this article are

—a discussion of concepts and definitions regarding IS including a theoretical model of elements that constitute IS;
—a classification framework for IS programs and projects as well as the application of the framework to known instances; and
—qualitative models summarizing reported IS benefits and adoption challenges.

The remainder of this article is structured as follows: Section 2 discusses the research methods we used for literature selection and analysis. Section 3 discusses definitions and concepts regarding IS and introduces a model describing the elements that constitute IS. Section 4 introduces a classification framework for IS and applies the framework to known IS programs and projects to demonstrate its capabilities. Section 5 introduces a qualitative model of seven IS benefits we synthesized from literature and case study reports. Section 6 presents a similar qualitative model describing adoption challenges. Section 7 suggests a roadmap for future research and closes the article with a conclusion.

1.1. Related Work

Naturally, this survey article contains, summarizes, and rearranges 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.

Elements of Inner Source. Sharma et al. [2002] analyzed open source 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 and tools*, and *organization and 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 (e.g., "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 3 we will present a theoretical model closing this gap.

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 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 4.

Benefits and Challenges of IS Adoption. Most of the surveyed publications reported benefits and challenges of IS adoption in the context of their case organizations. However, we identified two publications that focused on benefits and challenges.

Stol et al. [2011] developed a theory of IS adoption challenges from open source literature and a case study at an organization that adopted IS. They identified problems regarding component selection, documentation, support and maintenance, integration and architecture, and migration and usage.

Riehle et al. [2015] performed a multiple-case case study at three organizations adopting IS in a software product line setup. Riehle et al. [2015] found improved collaboration and knowledge sharing in IS that benefits component providers and reusers and finally results in improved development efficiency. They presented a model of adoption challenges that describes problems with developers, product unit managers, and expected problems with component quality, pilot projects, and development processes.

Our models of IS benefits (Section 5) and challenges of IS adoption (Section 6) summarize the findings of all the surveyed literature including Stol et al. [2011] and Riehle et al. [2015] and are of higher generality.

2. RESEARCH METHOD

We compiled this survey in two phases. 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.

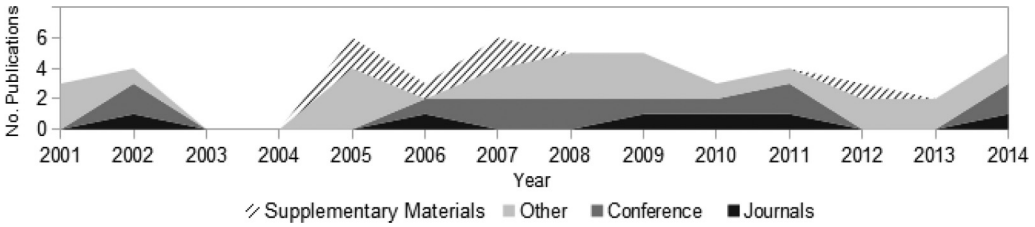

Fig. 1. Inner source related publications over time.

2.1. Literature Selection

For identifying the relevant literature, we searched for literature by phrases using Google Scholar, the ACM digital library, and IEEE Xplore. We utilized a variety of phrases and combinations including the following:

—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. As the body of literature regarding IS is smaller compared to more established development methods, it was not necessary to define more specific keywords. Subsequently, we manually determined whether a publication addresses IS by reading abstracts or the full publication. For the publications that were found to discuss IS, we searched for other publications of the same authors. We performed "snowballing" (transitively checking forward and backward citations) until the population of literature was exhausted. 43 publications were found that discuss IS. Iteratively, we classified the literature based on the discussed IS programs (with one paper possibly discussing more than one IS program). Appendix C presents an overview of the classified literature.

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 nonscientific articles (blog entries, magazines) were cited by the publications or found relevant for this survey. Figure 1 shows the number of IS publications in journals, conferences, and other venues per year. For completeness, it also shows nonscientific 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 shows the amount of literature regarding each organization ordered by the amount of scientific publications. Supplementary material and nonscientific 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 case studies were performed. Stol et al. [2014] performed

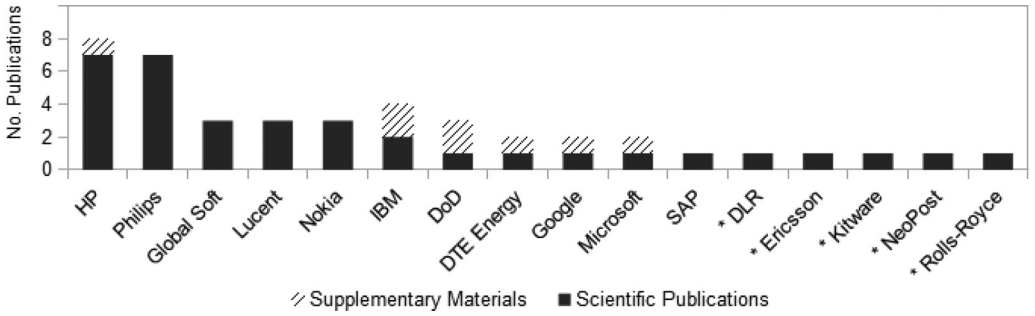

Fig. 2. Inner source related publications by organization.

an assessment of these demonstrating nine factors of IS adoption they have identified. Regarding Ericsson, Torkar et al. [2011] only analyzed the process alignment of Ericsson development and open source processes in preparation of IS adoption. An overview of the surveyed literature and the known IS programs are given in Appendix C. For further analyses we focus on the organizations (and their respective IS programs) that are not marked with an asterisk. Table I summarizes the resulting 13 IS programs that we considered for the survey.

2.3. Literature Analysis

Elements of Inner Source. One of this article's contributions 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[1] 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. The left side of Figure 3 shows an excerpt of an earlier iteration of the code system. We transferred this code system into our model of elements of IS (Section 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 article 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

[1]MaxQDA is a proprietary software tool that supports the coding process in qualitative data analysis. Further information on MaxQDA can be found at http://www.maxqda.com.

Table I. Overview of Organizations Using IS

Org.	Program Name	Summary
DTE Energy	-	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 Brown 2007]
Google	-	All source code at Google shares a single repository. All developers have read access and are encouraged to contribute [Whittacker 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 [Unknown 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 article we will only discuss inner source but not partner source.
	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 an IS 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 [Fox 2007; Vitharana et al. 2010].
	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	-	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 R&D 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].
Philips		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].
US DoD	Forge.mil	Forge.mil is a software forge and IS program used by the United States Department of Defense (US DoD) to collaborate internally and with their partners. Forge.mil allows collaboration with a broader audience within the US DoD or with partners outside the organization [Martin and Lippold 2011]. It is therefore an instance of inner source and partner source.
GlobalSoft		GlobalSoft adopted IS 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.

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 4.

Benefits and Challenges of IS Adoption. For developing models of IS benefits (Section 5) and challenges of IS adoption (Section 6) we identified benefits and challenges reported in the surveyed literature and arranged them into groups. Similar to our analyses regarding the elements of IS, the coding process delivered a hierarchical code system regarding benefits and adoption challenges. From the code systems

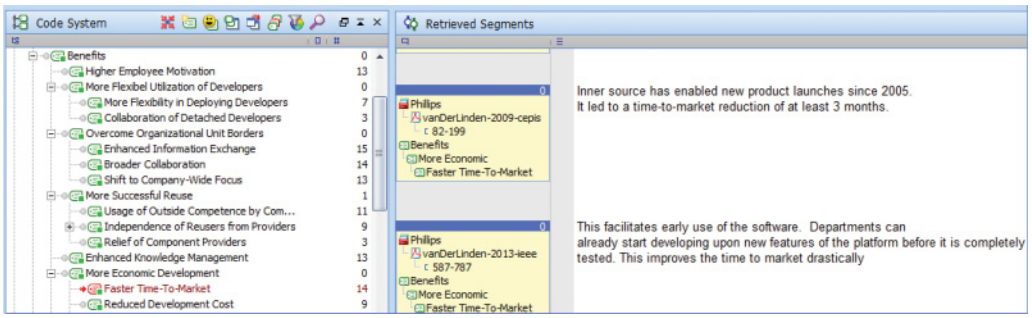

Fig. 3. MaxQDA screenshot with excerpt of an early iteration of our code system regarding text segments.

we derived the qualitative models of IS benefits and adoption challenges. Figure 3 shows an excerpt of an earlier iteration of the resulting code system in MaxQDA. The hierarchical code system can be seen on the left side of the image. The right side shows all text segments in the surveyed literature that discuss a selected benefit (in this example *faster time-to-market*).

3. IS DEFINITIONS AND ELEMENTS

Multiple organizations are using IS as part of their software development approach; researchers performed studies regarding the phenomenon. The authors of the respective publications each based their studies on specific definitions of IS. A selection of these definitions is presented in Appendix A.

3.1. IS Definitions

Dinkelacker et al. [2002][2] define IS as "the application of Open Source approach and benefits to developers within the corporate environment." The majority of definitions in literature [Goldman and Gabriel 2005; Melian 2007; Wesselius 2008; Gaughan et al. 2009; Gurbani et al. 2010; Riehle and Kips 2012; Stol et al. 2014] are akin to this first definition published by Dinkelacker et al. [2002] and share two characteristics:

(1) IS leverages practices from open source development.
(2) Contrary to open source, only a limited group of developers (employees of a specific organization) can take part in the community.

The "software product [or component] that is developed within an IS context" is called Inner Source Software (ISS) [Stol et al. 2011].

The surveyed literature distinguishes two units of analysis to analyze the IS phenomenon: IS programs and projects. We define an IS program 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. Other examples for IS programs are IBM's internal open source bazaar and community source [Fox 2007; Vitharana et al. 2010], Nokia's iSource [Lindman et al. 2008], Microsoft's CodeBox [Microsoft 2008], Google's common software repository [Whittaker et al. 2012], or the firm-internal open source program around SAP forge [Riehle et al. 2009].

[2]Dinkelacker and Garg [2001] is a workshop paper with similar content as Dinkelacker et al. [2002]. We omit citing the earlier version in the remainder of this article.

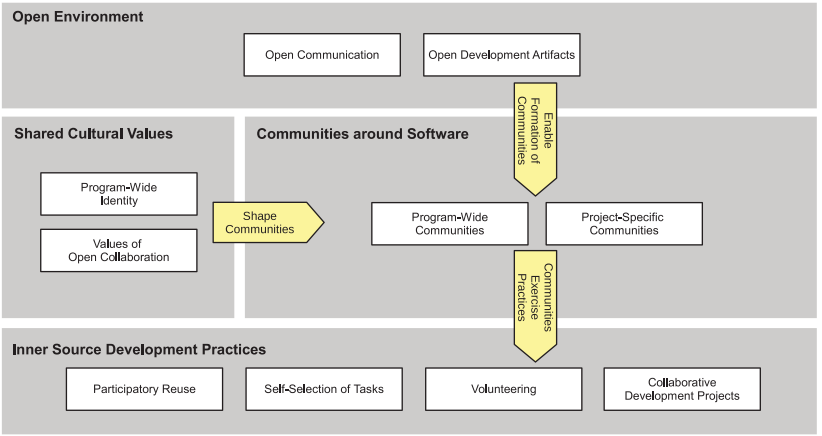

Fig. 4. Theoretical model of IS elements.

We define an IS project as follows:

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

An IS project does not have a fixed duration that is set prior to the project's start. Instead, IS projects are more akin to open source projects that do not have a defined end date. As in open source, the name of the project is often also used to address the ISS component. Example IS projects are listed in Appendix B.

The collection of all ISS components that are developed within the projects of an IS program form its *IS portfolio*, which we define as follows:

An IS portfolio is the set of all ISS 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. Contrarily, a project-specific view is focused on the surroundings of one specific project, the involved parties, and their interests. Distinguishing between these perspectives but also considering both is crucial for understanding an organization's IS efforts.

Within this article, we will use the verb form *to inner source* as follows: To innersource a software component means to either develop or maintain a software component using IS or to transition a software component such that it becomes an ISS component.

3.2. Theoretical Model of IS Elements

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 model characterizing IS based on four elements that have been extracted from the surveyed literature.

Figure 4 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. The gray areas are the four elements of IS (top-level categories of our 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 development practices* are exercised by a project-specific or program-wide community. Such practices can be observed in the open source world as well.

The arrows in Figure 4 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 development practices.

3.2.1. Open Environment. IS embraces an open environment. We found this open environment to be characterized by open communication, open development artifacts, and openness of contribution. 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 open source, 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 et al. [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 open source, 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 focusing 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 open source, encourages early and frequent releases of new incremental versions of ISS 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 2005]. 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 themselves but also the work performed on them is publicized. This enables individuals to infer other individuals' and organizational units' goals [Dabbish et al. 2012].

Secondly, in addition to reading and using open development artifacts, developers can contribute changes toward code and documentation even if the project is outside of their organizational 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 ISS component [Gurbani et al. 2010; van der Linden 2013]. As in open source, 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].[3] Once a patch has been accepted, it is the IS project's responsibility to maintain this portion of the code. As in open source, 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].

3.2.2. 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. We observed that 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 organizational 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 organizational 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.

[3]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 article.

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, organizational units, and project communities a higher degree of self-organization Riehle et al. [2009].

3.2.3. 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 ISS approach essentially establish an OSS [open source] 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 open source 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 article, we do not aim to resolve this deficiency. For the remainder of this article, we define a community broadly as an informal organization of individuals that communicate and collaborate with each other. In IS, these communities cross organizational unit boundaries.

Tamburri et al. [2013] identified 13 different types of communities and social networks relevant in a software engineering context. We suggest future research to identify which of these types project-specific and program-wide IS communities represent. We encourage researchers to evaluate how applicable community management practices of the identified community types are to communities in IS.

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 open source world (e.g., the communities around specific open source projects like JUnit or Libre Office).

3.2.4. IS Development Practices. Based on the surveyed literature, we identified four IS development practices (participatory reuse, self-selection of tasks, volunteering, and collaborative development projects). The existence of all IS development practices is not a necessary condition for a development endeavor to be called IS. We believe that more than the identified IS development practices can be implemented in IS programs. Future research will have to identify further development practices by observing more IS programs or finding potential new practices from the open source world.

Participatory Reuse. Participatory reuse is a form of software reuse in which reusers participate in developing and maintaining the software they reuse [Vitharana et al. 2010]. The term was coined by Vitharana et al. 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 contribute to the software they reuse [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 reuse 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 open source world. Wesselius [2008] summarizes: "In [open source], 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 and Breen 2007; 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 and Breen 2007; 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 development practices is based on developers contributing code to an IS project via patches. These can either be rejected or accepted by the owners of an ISS component. In open source, 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, for example, based on contributing patches, can occur in such a project [Senyard and Michlmayr 2004].

The surveyed literature reported on similar practices in IS. Organizations performed collaborative development projects in which they joined resources from different organizational units to develop an ISS component these organizational 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

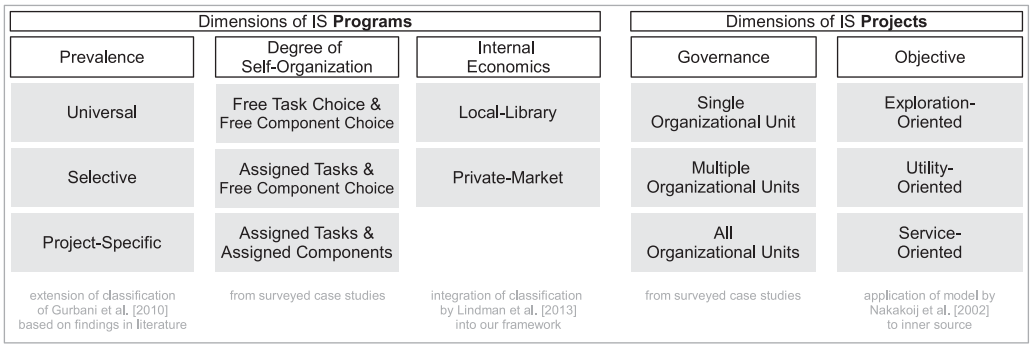

Fig. 5. Classification framework of IS programs and projects.

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 exercising other IS development practices.

4. CLASSIFICATION FRAMEWORK FOR IS

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, and internal economics). Our classification framework for IS programs delivers a multilabel 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 and objective).

Figure 5 gives 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.

4.1. Classification of IS 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 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. Figure 5 at the beginning of the paragraph also gives an overview of the classification framework and the classes of IS programs for each dimension.

4.1.1. 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 that 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 organizational 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].

Based on the surveyed case studies 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 an IS program's prevalence:

—Universal: All of the organization's software artifacts are publicized as an ISS component. There is no software component that is not inner-sourced.
—Selective: Only selected software artifacts are publicized as ISS. The remaining software components are not inner-sourced. Consequently, many IS projects exist.
—Project-Specific [Gurbani et al. 2010]: The majority of software components are not inner-sourced. Only a specific IS project is run within the IS program.

Of the three presented variants, implementing a 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 ISS and components are only excluded from the IS portfolio on a strictly exceptional basis.

4.1.2. 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 ISS 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 everyday tasks. Consequently, they not only contribute to ISS components used as part of their assigned everyday work. They are also enabled to contribute to ISS 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 ISS components to reuse for finishing their assigned tasks. No corset forces an individual to reuse one specific ISS component.
—Assigned Tasks and Assigned Components: The elements of IS (see Section 3) are implemented. However, no other freedoms are granted to individuals. They have no

right to freely choose which ISS components to reuse as part of their work. The used components are predefined by an existing corset (e.g., a software product line setup).

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

4.1.3. 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 ISS 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."

4.2. Classification of IS Projects

We classify IS **projects** based on two dimensions: The governance dimension describes who is responsible for the project and the developed ISS 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 open source world [Nakakoji et al. 2002] into our framework.

4.2.1. Governance. Governance of IS projects and regard of ISS components were 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 organizational unit*, *multiple organizational units*, and *all organizational units*.

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

Both governance by a single or by a multiple organizational unit require an explicit proclamation of responsible organizational units for the outcomes of an IS project. We believe collaborative development projects as described in Section 3 to typically result in governance of an IS project being executed by multiple organizational units.

Gurbani et al. [2010] reported on roles they implemented to enable the governance of a IS project governed by multiple organizational 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 ISS 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 organizational units but by all organizational units equally. Whittaker et al. [2012] uses the term "shared" components or libraries to refer to ISS components resulting from such projects. The surveyed literature did not present examples of IS projects governed by all organizational 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 such projects exist [Whittaker et al. 2012]. While the governance of these projects is independent of particular organizational 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 test the 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 organizational 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.

4.2.2. Objective. IS projects described in literature served different objectives. In the open source 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 accessible 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 a 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 ISS software [Nakakoji et al. 2002]. Service-oriented projects typically produce business critical ISS software components, have high quality requirements, and are conservative against rapid changes [Nakakoji et al. 2002].

An open source project's objective can change during its lifecycle [Nakakoji et al. 2002]. As in open source, 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

Table II. Classification of IS Programs

Degree of Self-Organization	Prevalence: Project-Specific	Prevalence: Selective	Prevalence: Universal
Assigned Tasks & Assigned Comp.	• GlobalSoft / SoftCom • Philips		
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			• Google

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].

4.3. Application of the 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. The application of the framework serves as a demonstration of its capabilities.

4.3.1. Classification of IS Programs. We classified 12 of the IS programs introduced in Section 2 according to our classification framework for IS programs. We first apply our classification framework of IS programs and subsequently lay out which of the IS development practices were implemented.

By Using the Classification Framework. Table II 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. Table II does not show the market-mechanisms dimension as 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 come close to a private-market as organizational units were required to pay a fee for reusing a component [Wesselius 2008]. However, contributions from outsiders neither occurred nor were they considered by the market-mechanisms in this early phase.

Regarding the degree of self-organization, we observed IS programs with assigned tasks and components (two), assigned tasks and free component choice (nine), and free task and component choice (one). IS programs with assigned tasks and components can be found at Philips [van der Linden 2009, 2013; van der Linden et al. 2009] and GlobalSoft [Höst et al. 2014] who use IS in the context of their software product line development. IS programs with free component choice but assigned tasks were implemented at SAP [Riehle et al. 2009], Microsoft [2008], and other organizations (see Section 4). 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 and Breen 2007]. While Google employees may use it to kick off a new project, it is also used for contributing to IS projects [Google-Blog 2006].

In the prevalence dimension, we observed selective (eight), project-specific (three), and universal (two) 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 (using a specific source code management system) [Lindman et al. 2010, 2013], the iSource program could become a universal IS program gradually. For the department of defense's Forge.mil program [Martin and Lippold 2011] we were not able to determine which degree of self-organization and internal economics they implement.

Table II shows that most of the surveyed organizations implemented selective (eight), and local-library (10) IS programs with assigned tasks and free component choice (nine). IS programs with assigned tasks and components (two) or free task choice (one) as well as universal IS programs (two) have a low prevalence.

Despite the few instances, due to the high prevalence of static dependencies between organizational units in the software industry (e.g., due to product line engineering), we assume IS programs with assigned tasks and components to be of high relevance for the software industry. At least one additional organization (Siemens) that implemented software product lines indicated that IS can benefit their development [Bartholdt and Becker 2012]. We hypothesize that the low prevalence of IS programs with free task choice is related to the unclear benefits and absence of proven steering mechanisms for such IS programs. We suggest future research on benefits and steering mechanisms.

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 software product line development and mitigate challenges in requirements elicitation, feature prioritization, and bottlenecks in the platform organizational units. However, IS programs with free component choice can be project-specific as well. Lucent only developed one ISS component. Still, there are no reports that this component was imposed on the developers. Organizational units were able to decide for themselves whether or not to use the ISS component [Gurbani et al. 2006, 2010].

Based on the information in the surveyed literature, we were only able to classify a small amount of IS programs (12). This leads to two problems. First, we were not able to identify IS programs for all possible combinations of classes. Consequently, Table II contains empty cells. However, we did find no indication that the combinations with no example IS program are invalid. Secondly, 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

Table III. Exercised IS Development Practice

IS Program	Collaborative Development Practice			
	Volunteering	Self-Selection of Tasks	Participatory Reuse	Collaborative Development
GlobalSoft / SoftCom			✓	✓
Philips			✓	✓
Lucent	✓		✓	
HP (CDP)			✓	✓
HP (CS)	unkown	unkown	unkown	unkown
IBM (CMS)			✓	
IBM (IIOSB)	unkown	unkown	unkown	unkown
Microsoft (CodeBox)			✓	
Nokia			✓	✓
SAP	✓			
DTE Energy			✓	✓
Google	✓	✓	(probably)	

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.

By IS Development Practices Exercised. IS programs not only differ on the dimensions of our classification framework but also by the IS development practices that are exercised within the programs. Table III summarizes the IS development practices exercised within 13 IS programs. Each row represents an IS program. Each column represents one development practice. In a case in which a development practice is reported to be exercised within an IS program, the specific cell is marked with a check mark. In a case in which a cell is not marked, it does not necessarily mean that the specific development practice is not exercised within the IS program. It solely means that literature regarding the IS program did not report about the practice. We were not able to infer sufficient information on the practices exercised at IBM's Internal Open Source Bazaar [Fox 2007], HP's Corporate Source [Dinkelacker et al. 2002], and Forge.mil [Martin and Lippold 2011]. The literature regarding these three IS programs did not report enough specifics to allow us to identify practices.

The individuals within the considered 10 IS programs exercised participatory reuse (nine) and collaborative development projects (five). Also, we observed volunteering (three) and self-selection of tasks (one).

Participatory reuse is the most prevalent development practice (8/10). Only Google and SAP do not report about participatory reuse. Giving the size of the Google organization and the fact that they implement universal IS, we conclude that participatory reuse is exercised with a high likelihood. None of the considered literature reported

Table IV. Classification of IS 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			• GlobalSoft (No name) • Lucent (SIP Server, later) • Philips (Medical Image SPL)
	All Organizational Units			

about collaborative development projects in the context of IS programs with free choice of tasks and components. We acknowledge the strong differences between these practices but do not assume them to be exclusive.

Initially, we were assuming volunteering and self-selection of tasks to only happen in universal or selective IS programs. This assumption is invalidated by the findings of Gurbani et al. [2006] who observed volunteering at Lucent's project-specific IS program.

4.3.2. Classification of IS Projects. We identified six IS projects with sufficiently detailed reports. For one of these projects (Lucent [Gurbani et al. 2010]) at least two distinct phases can be extracted and are listed separately. For the other nine IS programs, literature did not report about specific projects. Table IV summarizes the classification of the six considered IS projects. The table's construction is similar to Table II.

Regarding the governance dimension, literature reported projects governed by a single organizational unit (four) and multiple organizational units (two). A specific project without a governing organizational 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 organizational units and the implications for software quality and governance.

Regarding the objectives, the surveyed literature reported service-oriented (four), exploration-oriented (two), and utility-oriented (one) projects. Nakakoji et al. [2002] discussed that exploration- and utility-oriented open source projects are typically governed by a single organizational unit and evolve into service-oriented projects owned by multiple organizational 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 organizational units after becoming a service-oriented IS project. Microsoft's Code Box project discussed the need to empower the contributors

Table V. Seven Benefits of IS

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	

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].

Based on the information in the surveyed literature, we were only able to classify a small amount of IS projects (six). Consequently, Table IV contains empty cells as we were not able to identify example IS projects for every possible combination of IS classes. We found no indication in the surveyed literature that combinations without an example IS project are invalid.

The surveyed IS projects indicated a relationship between the class of an IS project and the exercised development practices that are common in the IS program hosting it: The development practices 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 (three/four). Gurbani et al. [2010] describe that focusing IS efforts on one specific project is a beneficial way to deal with critical assets. Also, all IS programs in the surveyed literature with assigned tasks and components implemented service-oriented projects. We suggest further research to find and classify additional IS projects and search for correlations between classes.

5. BENEFITS OF IS

We analyzed the surveyed literature following the method described in Section 2 to develop a model of the benefits of IS. Literature reported about a variety of benefits that IS has compared to traditional development methods. Table V gives an overview of the seven identified benefits (top-level categories of our code system). Six of these benefits aggregate more fine-grained benefits (lower-level categories).

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 IS. Riehle et al. [2009] performed a survey with SAP developers to validate that IS helped them to overcome intraorganizational boundaries. However, most of the surveyed literature neither validated the observed IS benefits nor discussed their generalizability. IS benefits should be treated as observations, 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.

5.1. 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. 2015].

Faster Time-to-Market. IS enabled organizations to achieve a faster time to market. Dinkelacker et al. [2002] and Riehle et al. [2015] describe "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."

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.

5.2. Overcoming of Organizational Unit Boundaries

Boundaries between organizational units can become hard to cross in large organizations. By creating an intraorganizational 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 organizational units' work [Riehle et al. 2009]. IS facilitates an improved organization-wide perspective and intraorganizational collaboration [Riehle et al. 2015].

Cost and Risk Sharing Among Organizational Units. IS strengthens an organization-wide focus by promoting cost and risk sharing between organizational 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 result can be increased trust between organizational units.

Cost and risk sharing between organizational units can be achieved with collaborative development projects where organizational units jointly incubate IS projects. Each organizational unit supplies a fraction of the resources needed for the project. Consequently, cost and risk are shared among the organizational units.

Collaboration across Organizational Unit Boundaries. IS enables collaboration among the organizational unit boundaries to a degree not possible in traditional setups [Vitharana et al. 2010; van der Linden 2013]. Collaborations among organizational units' boundaries are more flexible in IS, enabling one 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.

Program-Wide Information Exchange. Easy access to information spread over the organization is one of the main principles of IS [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 projects 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].

5.3. More Successful Reuse

The surveyed literature reported IS' 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.

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 can translate into higher quality components and enables reusers to make ISS components fitter for their use cases by contributing to them.

Independence between Reusers and Providers. In a traditional 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 the efficiency benefits of IS [Gurbani et al. 2006, 2010].

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."

5.4. Better Software Product

Organizations reported that IS enabled them to achieve a higher quality software product than with traditional development methods alone.

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 also has 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 quasipublic scrutiny to make developers "feel compelled to strive for high quality of contributions." Finally, the increased employee motivation can result in higher code quality [Riehle et al. 2009].

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."

Also, 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 scope of their everyday work.

5.5. More Flexible Utilization of Developers

Riehle et al. [2015] observe that in IS developers can be used more flexibly leading to improved resource management.

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, the projects typically share a similar infrastructure and developer switching projects benefit from the openly available project information. In a project-specific IS program, only one IS project is executed. The ease of deployment between the potentially many non-IS projects does not change due to IS adoption.

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.

5.6. 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 intraorganizational knowledge sharing [Riehle et al. 2015].

Selective or universal IS programs make multiple IS projects and their documentation openly available. Consequently, such IS programs also make available a broader portfolio of knowledge. 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].

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].

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.

5.7. Higher Employee Motivation

IS can facilitate higher motivation and job satisfaction of developers [Riehle et al. 2015] and lead to "improved [. . .] morale and retention" [Martin and Aitken 2012]. The increased motivation can result in the development practice volunteering where developers are intrinsically motivated similar to developers in open source are [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 developers are generally motivated by IS programs with a high degree of self-organization (free task choice and free component choice).

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.

6. CHALLENGES OF IS ADOPTION

Within this section, we present a qualitative model of challenges of IS adoption. In the surveyed literature, no direct link between adoption challenges and specific IS benefits

Table VI. Challenges of IS Adoption

Mismatch of inner source and existing organizational setup	
Resistance & significant change • Significant change in working style • Cultural unfitness • Resistance due to individual disadvantages	**Diversity among organizational units** • Diversity of processes • Diversity of tools
Local interests of organizational units • Fear of resource loss • Mismatch of perceived and actual code ownership • Fear of maintenance effort	
Issues with inner source adoption itself	
Difficult utilization of openness • Navigate through large amount of data • Unawareness of ongoing work • Complexity of understanding foreign code	**Application of control & steering** • Implementation of new leadership style • Insufficient models & metrics • Dysfunctional incentive systems
Resentments against code transparency • Fear of security problems • Fear of intellectual property loss • Resentments against scrutiny	**Contribution process not running smoothly** • Unfit contributions • Uncontrolled forking • Formation of patch queues
What to inner-source?	

was evident. To the contrary, we believe that the identified adoption challenges affect all IS benefits as they can affect the IS adoption altogether.

The model of challenges of IS adoption was inferred in the same process as the model on benefits of IS. Table VI summarizes our model of adoption challenges. We identified a total of eight adoption challenges (top-level categories in the code system). Seven of them are characterized further by more fine-grained challenges (lower-level categories). We found that three of the challenges are due to the mismatch of IS practices and the traditional preexisting organizational setup (resistance and significant change, diversity among organizational units, local interests of organizational units). The remaining four challenges are inherent to IS adoption itself.

6.1. Mismatch of IS and Existing Organizational Setup

6.1.1. Resistance and Significant Change. The adoption of IS induces significant change to the organization and challenges existing cultural values, established beliefs, and heuristics [Neus and Scherf 2005]. Consequently, it can trigger resistance [Neus and Scherf 2005] or simply reluctance to contribute [Stol et al. 2011].

Significant Change in Working Style. IS is different from traditional development methods. Riehle et al. [2009] of SAP describe that "many open source best practices fly in the face of traditional software development methods [. . .]. [They] don't view users as customers [. . .], but rather they empower users to become co-developers." The software components rely on incremental collaborative improvement [Neus and Scherf 2005]. Consequently, far reaching changes to the processes, roles, and other artifacts of the organizational setups are necessary. Such organizational changes can be challenging.

Cultural Unfitness. An organization's culture can conflict with the cultural values inherent to IS and trigger resistance [Neus and Scherf 2005]. Management cannot simply force this culture to change [Wesselius 2008] and needs to apply well-coordinated means to adjust the cultural system of an organization [Neus and Scherf 2005; Wesselius 2008]. The implementation of an IS program requires a "mindset shift from

delivering [a] final product to incremental quality code" and can lead to a "culture shock and dissonance" [Martin and Aitken 2012]. IS conflicts with strictly hierarchical cultures [Riehle et al. 2015].

Some organizations' cultures are less open to change. We believe universal IS programs and IS programs with a high degree of self-organization (free choice of tasks) as well as IS projects owned by all organizational units to be particularly challenging for these organizations. Such IS programs and projects introduce most change and are significantly different than traditional development practices.

Resistance Due to Individual Disadvantages. Parties that fear disadvantages resulting from IS adoption are likely to resist [Neus and Scherf 2005; Riehle et al. 2015]. To make individuals participate in the IS program, it is crucial to have them understand the benefits they can get out of it. The question "What's in it for me?" must be answered [Neus and Scherf 2005].

6.1.2. Diversity among Organizational Units. Diversity among organizational units challenges collaboration among them and consequently IS adoption.

In project-specific IS programs, diversity of processes and tools can become challenging, because contributors need to adapt to idiosyncratic conditions at the one existing IS project. In selective or universal IS programs, the diversity becomes more challenging as there are multiple IS projects. Each of these IS projects potentially have their own idiosyncratic processes and tools.

Diversity of Processes. Among an organization's organizational units often a "lack of process consistency" can be observed [Martin and Aitken 2012]. This hinders collaboration and the utilization of open information and infrastructure in practice. Gurbani et al. [2010] of Lucent observed that "each business division has idiosyncratic processes for feature creation and prioritization that must be accommodated." In regulated environments such as medical software diverse processes cannot easily be changed and unified [van der Linden 2013].

Diversity of Tools. Similar to diverse processes, diversity of tools challenges IS adoption. SAP [Riehle et al. 2009], HP [Dinkelacker et al. 2002], and Lucent [Gurbani et al. 2010] faced challenges due to the diversity of software development tools used in different organizational units. Incompatible tools include source code management, and bug reporting tools [Dinkelacker et al. 2002]. At Lucent, the diversity of tools lead organizational units to fork the IS project for the sole reason of being able to use the infrastructure they were used to [Gurbani et al. 2010].

6.1.3. Local Interests of Organizational Units. Middle managers leading organizational units have their own interests that may conflict with the organization-wide perspective necessary for IS adoption. We found that they fear loss of resources and an increased maintenance effort. Also, a mismatch between perceived and actual code ownership can occur leading to conflicts among organizational units.

Fear of Resource Loss. Middle managers fear that IS may result in disadvantage for their organizational unit. Dinkelacker et al. [2002] of HP observed that "It's also possible that some managers or even developers may be inimical to contributing any resources to perceived resource competitors within the organization." If the middle managers' performance goals depend on the completion of certain tasks, losing resources to IS projects can become a personal disadvantage as well. Conversations with IS practitioners and consultants indicate that this challenge is of high importance and has a large impact on the IS program's success: Middle managers have means to effectively hinder individuals from contributing to IS.

Implementing private-market IS programs can mitigate the middle managers' fear to lose resources. Private-markets can be designed to reimburse for both providing an ISS component or contributing to it. However, as discussed in Section 4, private-markets can also hinder collaboration.

Mismatch of Perceived and Actual Code Ownership. IS allows individuals or organizational units to contribute significant amounts of code to other organizational units' code. Vitharana et al. [2010] discuss that this can lead to tension:

> "While those consumers who simply use the software ex post might not claim a stake in the software, those potential consumers who participate in various aspects of the IOS [internal open source] project (e.g., make change requests) could consider themselves part owners. [. . .] With possible equity in influence, the two parties could conceivably perceive themselves to be equal owners of the software."

Vitharana et al. [2010] suggest further research into the tension between actual and perceived ownership. Resulting conflicts can hinder further adoption and the success of IS. We assume unclear rules on ownership can hinder organizational units from both contributing to and providing an ISS component because of the increased uncertainty regarding risks and benefits.

We assume a mismatch of perceived and actual code ownership to be less likely when the ownership of an IS project is explicitly defined (the project is owned by one or multiple organizational units).

Fear of Maintenance Effort. Usually owners of an IS project are responsible for maintaining the developed ISS component [Stol et al. 2011]. Each portion of code contributed to an ISS component adds to the code base that needs to be maintained. Stol et al. [2011] observed that owners of IS projects were reluctant to accept contributions because they were afraid of additional maintenance effort.

6.2. Issues with IS Adoption Itself

6.2.1. Difficult Utilization of Openness. Openness is one of the elements that constitute IS. Establishing openness of code, documentation, and communication can result in a large amount of information. Consequently, information overload can occur. Properly utilizing openness while avoiding information overload is a challenge of IS adoption.

Navigation through Large Amount of Data. A portfolio of ISS components and the program-wide community can quickly grow to dimensions that are not easily digestible for a human mind. Lucent [Gurbani et al. 2006] and HP [Dinkelacker et al. 2002] found navigating through large code portfolios or other information catalogs challenging. Dinkelacker et al. [2002] summarize that "searching and navigating through the several projects and personnel details can easily become time consuming to the point of making the effort worthless."

Navigation through the open data is particularly challenging in selective and universal IS programs that run more than one IS project. However, also in project-specific IS programs a large amount of data is opened up and can become difficult to navigate.

Unawareness of Ongoing Work. IS enables organization-wide information exchange. However, developers within an IS program are often unaware of relevant work performed by others. Public availability of information regarding ongoing work is not always sufficient to raise awareness. Gurbani et al. [2006] of Lucent describe that a "significant coordination problem was knowing what kind of work was going on for the server," which can lead to redundant work.

Complexity of Understanding Foreign Code. Once an ISS component for reuse is identified, it can still be difficult for an individual to make use of it [Stol et al. 2011]. Understanding software source code of other authors is a nontrivial and complex task. It is crucial to bring developers up to speed on the ISS components' functionality and design [Gurbani et al. 2010]. Enabling developers to understand other developers' code is a challenge not only relevant in IS but in software engineering in general. IS' openness of documentation and development artifacts have the potential to mitigate this challenge.

6.2.2. Application of Control and Steering. IS gives more freedom to developers and other employees. This conflicts with traditional hierarchical application of control in organizations. Challenges in managing subordinates can be a consequence.

We believe IS programs with a high degree of self-organization (free task choice) to be specifically challenging. As developers choose a part of their tasks themselves, the control exercised by managers becomes less immediate.

Implementation of New Leadership Style. IS requires a leadership style that supports meritocracy and self-organization and functions similar to open source [Stol et al. 2014; Riehle et al. 2009]. However, IS is less self-organized than open source, because IS "cannot be fully self-organizing, as there are business aspects to consider such as the timely delivery of products that depend on the shared [IS] asset" [Stol and Fitzgerald 2015].

We found implementing such a leadership style (inheriting attributes from leadership in both open source and traditional organizations) to challenge IS adoption. It is not clear "how to move from a control based organization to an organization based on empowerment and trust" [Melian and Mähring 2008]. Hierarchical leadership structure hinders collaboration and can contradict with IS [Dinkelacker et al. 2002].

Insufficient Models and Metrics. IS is lacking models and metrics [Wesselius 2008; Riehle et al. 2015]. A set of metrics and business models have proven to be helpful in coping with open source. These models do not apply to IS as they mostly explain how organizations gain profit within an open source community [Wesselius 2008; Gurbani et al. 2006].

Gurbani et al. [2006] remark:

> "Additionally, there are challenges in comparing internally developed resources with commercially available ones since it is difficult to determine the actual cost of the internal software, or to measure the benefits, such as modifiability, that come from owning the code."

Such metrics would be necessary as development products and projects have to operate within budgets and justify their decision on whether to procure components from outside vendors.

However, some simple counting heuristics were applied to estimate an IS project's and program's success as well as a project's quality [Microsoft 2008; Dinkelacker et al. 2002; Riehle et al. 2009; Gurbani et al. 2006].

Dysfunctional Incentive System. We found that an organization's incentive system can discourage individuals from participating in IS. Melian et al. [2002] of HP observe:

> "The traditional hierarchical organizations reward and promote cohesive project or product related behavior. [. . .] While community-help and visibility is encouraged, it is not the main factor when considering the yearly progress of an employee or their managers. Helping out another person in a different group can sometimes be detrimental to an individuals career."

Incentive systems like the annual progress evaluation discussed by Melian et al. [2002] can hinder successful IS adoption, if in any form individuals are penalized for participating in IS. They can discourage individuals from participating in IS and mitigate the individually perceived IS benefits.

6.2.3. Resentments against Code Transparency. Openness and especially the transparency of code are elements of IS. But code transparency also triggers fears and potentially causes challenges.

Fear of Security Problems. Dinkelacker et al. [2002] of HP observed reluctance to opening code in projects with high security requirements. Schryen [2011] analyzed the effects open source development has on the produced software's security. They found no empirical evidence that open source development (and thus code transparency) is a primary driver for security vulnerabilities. Given the similarities between open source and IS, we assume their results are transferable to IS as well.

Fear of Intellectual Property Loss. Knowledge and software code are an important outcome of a software organization's work. The so-created intellectual property is an important resource and worth protecting. Stakeholders in organizations fear that IS may lead to the leak of intellectual property. Melian and Mähring [2008] describe that "the balance between openness and safeguarding of intellectual property places intellectual property decisions with individual developers and makes these decisions a part of daily practice." This induces a risk of wrong decisions if no clear policies are in place.

The risk of disclosing intellectual property also depends on the prevalence of IS. By potentially involving more developers, universal IS programs are at higher risk than selective or project-specific programs. We assume that also a higher degree of self-organization may impose a greater risk as it allows developers to get involved with more ISS components and thus potentially more code with sensitive intellectual property.

Resentments against Scrutiny. IS adoption goes hand in hand with openness of code and artifacts. The so-created scrutiny can lead to developer resistance as it creates a "virtual panopticon in which every mistake is likely to get noticed" and developers cannot hide behind a virtual network identity as in open source [Melian and Mähring 2008]. Developers have resentments against such scrutiny:

> "Most corporations today operate on a hierarchical organizational structure. This complicates the process of code sharing by having differing product road-maps and time-lines, where some managers may just push to get something delivered by a promised date, irrespective of code quality, which might then be an embarrassment to post into the [IS] code tree." (Dinkelacker et al. [2002] regarding HP)

> "People were erring on the side of keeping code and information closed and hidden on their own computers, rather than exposing themselves to potential criticism [. . .]" (Neus and Scherf [2005] regarding IBM)

The quotes indicate that some developers fear criticism regarding their code or embarrassment in front of their colleagues. The fear is worsened if in critical project situations developers feel forced to develop code they perceive to be of substandard quality.

6.2.4. Contribution Process not Running Smoothly. Implementing and running contribution processes presented challenges to the case organizations in the surveyed literature.

Unfit Contributions. Contributions do not always fit the ISS component they are being contributed to [Stol et al. 2011]. Factors that can disqualify a contribution for

acceptance are missing generality, insufficient code quality, or simply incompatibility with the component owners' plans.

To satisfy many reusers, component providers have an interest in keeping the generality of an ISS component high. Gurbani et al. [2006] observed that contributions often lacked generality and consequently challenged IS adoption: Developers "were unaccustomed to thinking and designing solutions that were more general than their own product line. They typically did not make changes to the SIP server in a way that would support all users [. . .], unconcerned about building in dependencies that limited the generality of their work." In addition to being unaccustomed to delivering general code, product specific time pressure can lead to reduced generality of contributions. Missing generality of contributions can lead to a less reusable ISS component and architectural erosion [Gurbani et al. 2006].

Uncontrolled Forking. IS allows component reusers to utilize and change an ISS component's source code at will. Being able to perform local changes to code is an intended benefit of IS [Stol et al. 2014]. However, it can also challenge the IS adoption if uncontrolled forking occurs. Lucent faced such uncontrolled forking [Gurbani et al. 2006, 2010]: Instead of contributing back to the IS project, different teams created their own forks that diverged from the original implementation. Thus, maintenance efforts needed to be performed redundantly. Lucent limited forking by evangelizing to the teams the benefits of using a shared common repository instead of maintaining their own forks. If forking cannot be avoided, the forked ISS components should be synchronized with the master periodically [Gurbani et al. 2006, 2010].

Formation of Patch Queues. Contributions also create additional workload for the component providers as they need to be properly reviewed and subsequently accepted or rejected. Gurbani et al. [2006] of Lucent observed that the IS project's benevolent dictator had insufficient time to review patches in a timely manner. A group of committers was created to support him. However, they were reluctant to set aside time. This can lead to a patch queue backlog: A long backlog of patches is piling up and the waiting time until a specific patch is accepted or rejected increases.

6.2.5. What to Inner-Source? It was not clear which software benefits most from being inner-sourced. In universal IS programs, no decision on suitable ISS components needs to be made because all software is inner-sourced by definition. For project-specific and selective IS programs, however, Gurbani et al. [2006] of Lucent summarize that "it is not clear, in general, [. . .] when to initiate a project that can serve as a shared [IS] resource." Identifying the right components to inner-source is specifically important in the early phases of an IS program. The failure or success of an IS pilot project can predetermine the success of the IS adoption [Stol et al. 2014; Stol and Fitzgerald 2015]. Software components with a modular architecture and multiple interested stakeholders are particularly fit to be inner-sourced [Stol et al. 2014].

7. CLOSING

7.1. Roadmap for Future Research

Our survey arranged research results about IS. We presented a model of IS elements, a classification framework for IS, IS benefits, and adoption challenges. Some areas that we believe are relevant for researchers and practitioners were not yet covered by the literature. We suggest the following future research:

Find and Classify IS Development Practices. As part of the elements of IS, we identified four IS development practices that were exercised within the surveyed IS programs. We doubt their completeness. We believe many collaborative practices can be

observed in the open source world. We suggest future research to identify such additional practices from open source and evaluate their fitness for organization internal use. We envision a pattern system of IS practices similar to the design patterns by Gamma et al. [1994].

Understand IS Programs with Free Task Choice. In IS programs with free choice of tasks, developers are granted much autonomy. Only a few publications regarding this class of IS exist. We suggest further explorative research to identify what the benefits and challenges of IS programs with free task choice are and how they can be steered effectively.

Understand IS Projects without Governing Organizational Units. We discussed the existence of IS projects that were governed by all organizational units. We suggest further research on the experiences of organizations that implemented IS projects with such a governance model. It is unclear to us how conflicts can be resolved, which components are fit to be maintained without ownership by select organizational units, how software quality can be best ensured, and how software evolution can be steered in such projects.

Develop Models for Private-Market IS. Philips considered the implementation of private-market IS. We suggest further research on how models for private-markets must be designed and how they can benefit and hinder IS adoption.

Classify Additional IS Programs and Projects, Extend Classification Framework. Based on the surveyed literature, we were only able to classify a small amount of IS programs (12) and IS projects (six). We suggest finding and classifying additional IS programs and projects, and studying the prevalence of each class, and correlations between the classes. Consequently, researchers could draw conclusions on which classes work best together under what conditions. The classification framework we presented is developed from literature that was available to us at time of analysis. We believe that future research can find additional dimensions and classes for classifying IS programs and projects. We encourage researchers to extend our classification framework.

Evaluate IS Benefits. We synthesized a qualitative model of IS benefits from the surveyed literature. However, we discussed that the validity and specifically generalizability of IS benefits presented by the surveyed literature is unclear. We suggest future research into validating the reported IS benefits and the extent of their effect on adopting organizations.

Identify Community Management Practices. We were not able to infer proven best practices of community management in IS. We suggest future research to identify and evaluate community management practices. This can be done by mapping IS communities to the community types presented by Tamburri et al. [2013] and then identifying how management practices for these community types can be transferred to IS.

Metrics and Incentives Systems. IS is lacking metrics. We suggest future research to identify metrics to measure the progress of IS adoption and the quality of IS programs, projects, and communities. Such metrics could serve as the foundation for IS-specific incentive systems for developers and development managers.

Evolution of IS Programs and Projects. The early phases of IS (IS adoption) has been extensively covered by literature. However, it is not clear to us how programs as a whole evolve. Similarly, it is unclear to us how IS projects evolve. The reports regarding Philips [van der Linden et al. 2009; van der Linden 2009, 2013] discussed openness as one dimension of software evolution. We discussed possible similarities between the

evolution of open source projects as discussed by Nakakoji et al. [2002] and IS projects. We suggest future research on how IS programs and projects evolve.

Envision Tools. Today, development methods like agile or plan-driven development are supported by an extensive amount of software development tools. We suggest researchers look at tool requirements that emerge around IS and envision and validate potential solutions for these requirements.

7.2. Conclusion

The majority of scientific publications regarding IS describe the phenomena in the context of one or a few organizations. However, the research area lacked a systematic arrangement of prior research results. With this survey article, we presented an in-depth literature review and analysis to resolve this shortcoming by systematically arranging prior research results regarding IS.

We provided a holistic definition of IS and related concepts and provided a model of four elements that constitute IS. We introduced a classification framework for IS programs based on three dimensions and IS projects based on two dimensions. We applied our framework to demonstrate its capabilities and delivered a map of already researched IS programs and projects.

We combined known case study reports to synthesize qualitative models regarding IS benefits and challenges of IS adoption. These qualitative models deliver an aggregated overview of the experiences made in at least 13 IS programs.

ACKNOWLEDGMENTS

We thank Ann Barcomb, Hannes Dohrn, and Andreas Kaufmann for their constructive comments during the writer's workshop of this article as well as Klaas-Jan Stol, Sushil K. Bajracharya, and the anonymous peer reviewers for their feedback that helped us to improve this article.

REFERENCES

Pär J. Ågerfalk, Brian Fitzgerald, and Klaas-Jan Stol. 2015. *Software Sourcing in the Age of Open: Leveraging the Unknown Workforce*. Springer.

Gary Anthes. 2005. Software Reuse: Making it Work—DTE Energy may have cracked the cultural side of reusable software. (2005). Interview with Lynne Ellyn of DTE Energy. Last retrieved in February 2015, 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). Blog article. Last retrieved in February 2015, http://archive.oreilly.com/pub/post/microsoft_office_experiments_w.html.

Jörg Bartholdt and Detlef Becker. 2012. Scope extension of an existing product line. In *Proceedings of the 16th International Software Product Line Conference - Volume 1 (SPLC'12)*. ACM, New York, NY, 275–282. DOI: http://dx.doi.org/10.1145/2362536.2362573

Doug Beizer. 2009. DOD launches site to develop open-source software. *The Business of Federal Technology* (2009). Last retrieved in March 2015, http://fcw.com/articles/2009/01/30/dod-launches-site-to-develop-open-source-software.aspx.

Herbert H. Clark and Susan E. Brennan. 1991. *Grounding in Communication*. American Psychological Association, 127–149.

Kevin Crowston, Kangning Wei, James Howison, and Andrea Wiggins. 2008. Free/libre open-source software development: What we know and what we do not know. *ACM Comput. Surv.* 44, 2, Article 7 (March 2008), 35 pages. DOI: http://dx.doi.org/10.1145/2089125.2089127

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)*. ACM, New York, NY, 1277–1286. DOI: http://dx.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).

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

Steve Fox. 2007. IBM Internal Open Source Bazaar. (2007). Presentation at the IBM Linux Technology Center in November 2007.

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1994. *Design Patterns: Elements of Reusable Object-oriented Software*. Pearson Education.

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 *Proceedings of the 35th Euromicro Conference on Software Engineering and Advanced Applications (SEAA'09)*. 20–27. DOI:http://dx.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). Authored by Alex. K; Last retrieved in March 2015, http://googleblog.blogspot.de/2006/05/googles-20-percent-time-in-action.html.

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 5th Workshop on Open Source Software Engineering (5-WOSSE)*. ACM, New York, NY, 1–6. DOI:http://dx.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 *Proceedings of the 28th International Conference on Software Engineering (ICSE'06)*. ACM, New York, NY, 472–481. DOI:http://dx.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), 155–159. DOI:http://dx.doi.org/10.1145/1646353.1646392

Gary Hamel and Bill Breen. 2007. *The Future of Management*. Harvard Business School Press.

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

Rick Lehrbaum. 2001. HP launches "CoolBase" open source project. *LinuxGizmos* (2001). Last retrieved in March 2015, http://archive.linuxgizmos.com/hp-launches-coolbase-open-source-project/.

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. DOI:http://dx.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, 107–121. DOI:http://dx.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, 381–387. DOI:http://dx.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 *Proceedings of the 18th European Conference on Information Systems*.

Guy Martin and Andrew Aitken. 2012. Inner Sourcing—Community Development Practices in Corporate IT. (2012). Last retrieved in March 2015, https://www.blackducksoftware.com/resources/webinar/understanding-inner-source-fundamentals-transparency-collaboration-and-self-organization.

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, 334–337. DOI:http://dx.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. DOI:http://dx.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. Economic Research Institute, Stockholm School of Economics (EFI).

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.

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, 93–104. DOI:http://dx.doi.org/10.1007/978-0-387-09684-1_8

Microsoft. 2008. Open Source at Microsoft—Bringing the Open Source Approach In-House. (2008). White paper, last retrieved March 2015, http://download.microsoft.com/download/2/6/7/267E8B26-B94B-4BF6-88E8-32B3B3AF6F09/CodeBox_vfinal.pdf.

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)*. ACM, New York, NY, 76–85. DOI:http://dx.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 Syst. J.* 44, 2 (2005), 215–225. DOI:http://dx.doi.org/10.1147/sj.442.0215

Patrick Oor, René Krikhaar, and ICT NoviQ. 2008. Balancing technology, organization, and process in inner source. *Dagstuhl Workshop 08142: Combining the Advantages of Product Lines and Open Source* (2008), 1548.

Tim O'Reilly. 2000. Archived email discussion on Open Source and OpenGL. (2000). Last retrieved in February 2015, http://archive.oreilly.com/pub/a/oreilly/ask_tim/2000/opengl_1200.html.

Eric Raymond. 1999. The cathedral and the bazaar. *Knowl., Technol. Policy* 12, 3 (1999), 23–49.

Dirk Riehle. 2007. The economic motivation of open source software: Stakeholder perspectives. *Computer* 40, 4 (2007), 25–32.

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, 18–30. DOI:http://dx.doi.org/10.1007/978-3-642-03132-8_2

Dirk Riehle. 2015. The five stages of open source volunteering. In *Crowdsourcing*, Wei Li, Michael Huhn, and Wei-Tek Tsai (Eds.). Springer. Republished from The Five Stages of Open Source Volunteering. Friedrich-Alexander-Universität Erlangen-Nürnberg, Dept. of Computer Science, Technical Report, CS-2014-01, March 2014. Erlangen, Germany, 2014.

Dirk Riehle, Maximilian Capraro, Detlef Kips, and Lars Horn. 2015. *Inner Source in Platform-Based Product Engineering*. Technical Report CS-2015-02. Faculty of Engineering, Department of Computer Science, Open Source Research Group. 16 pages.

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 (2009), 52–58.

Dirk Riehle and Detlef Kips. 2012. *Geplanter Inner Source: Ein Weg zur Profit-Center-übergreifenden Wiederverwendung*. Technical Report CS-2012-05. Faculty of Engineering, Department of Computer Science, Open Source Research Group.

Jason E. Robbins. 2005. 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.). MIT Press, Cambridge, MA, 245–264.

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*. ACM, 46.

Guido Schryen. 2011. Is open source security a myth? *Commun. ACM* 54, 5 (May 2011), 130–140. DOI:http://dx.doi.org/10.1145/1941487.1941516

Anthony Senyard and Martin Michlmayr. 2004. How to have a successful free software project. In *Proceedings of the 11th Asia-Pacific Software Engineering Conference, 2004*. 84–91. DOI:http://dx.doi.org/10.1109/APSEC.2004.58

Srinarayan Sharma, Vijayan Sugumaran, and Balaji Rajagopalan. 2002. A framework for creating hybrid-open source software communities. *Inform. Syst. J.* 12, 1 (2002), 7–25.

Phillip Smith and Chris Garber-Brown. 2007. Traveling the open road: Using open source practices to transform our organization. In *Agile Conference (AGILE), 2007*. 156–161. DOI:http://dx.doi.org/ 10.1109/AGILE.2007.65

Mirjana Spasojevic and Tim Kindberg. 2001. Evaluating the cooltown user experience. In *Ubicomp 2001 Workshop: Evaluation Methodologies for Ubiquitous Computing*.

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. University of Limerick.

Klaas-Jan Stol, Paris Avgeriou, Muhammad Ali Babar, Yan Lucas, and Brian Fitzgerald. 2014. Key factors for adopting inner source. *ACM Trans. Softw. Eng. Methodol.* 23, 2, Article 18 (April 2014), 35 pages. DOI: http://dx.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. *Inf. Softw. Technol.* 53, 12 (2011), 1319–1336.

Klaas-Jan Stol and Brian Fitzgerald. 2015. Inner source-Adopting open source development practices in organizations: A tutorial. *IEEE Software* 32, 4 (July 2015), 60–67. DOI: http://dx.doi.org/10.1109/MS.2014.77

Damian A. Tamburri, Patricia Lago, and Hans van Vliet. 2013. Organizational social structures for software engineering. *ACM Comput. Surv. (CSUR)* 46, 1 (2013), 3.

David R. Thomas. 2006. A general inductive approach for analyzing qualitative evaluation data. *Am. J. Eval.* 27, 2 (June 2006), 237–246.

Richard Torkar, Pau Minoves, and Janina Garrigós. 2011. Adopting free/libre/open source software practices, techniques and methods for industrial use. *J. Assoc. Inf. Syst.* 12, 1 (2011), 88–122.

Frank van der Linden. 2009. Applying open source software principles in product lines. *Cepis Upgrade—Eur. J. Inf. Profess.* 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, 216–235. DOI: http://dx.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 (July 2009), 77–83. DOI: http://dx.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. *J. Manag. Inf. Syst.* 27, 2 (2010), 277–304.

Jacco Wesselius. 2008. The bazaar inside the cathedral: Business models for internal markets. *IEEE Software* 25, 3 (May 2008), 60–66. DOI: http://dx.doi.org/10.1109/MS.2008.79

James A. Whittaker, Jason Arbon, and Jeff Carollo. 2012. *How Google Tests Software*. Addison-Wesley.

David Worthington. 2005. IBM Turns to Open Source Development. (2005). Interview with Doug Heintzman of IBM. Last retrieved in February 2015, http://betanews.com/2005/06/13/ibm-turns-to-open-source-development/.

Received October 2015; revised April 2016; accepted September 2016