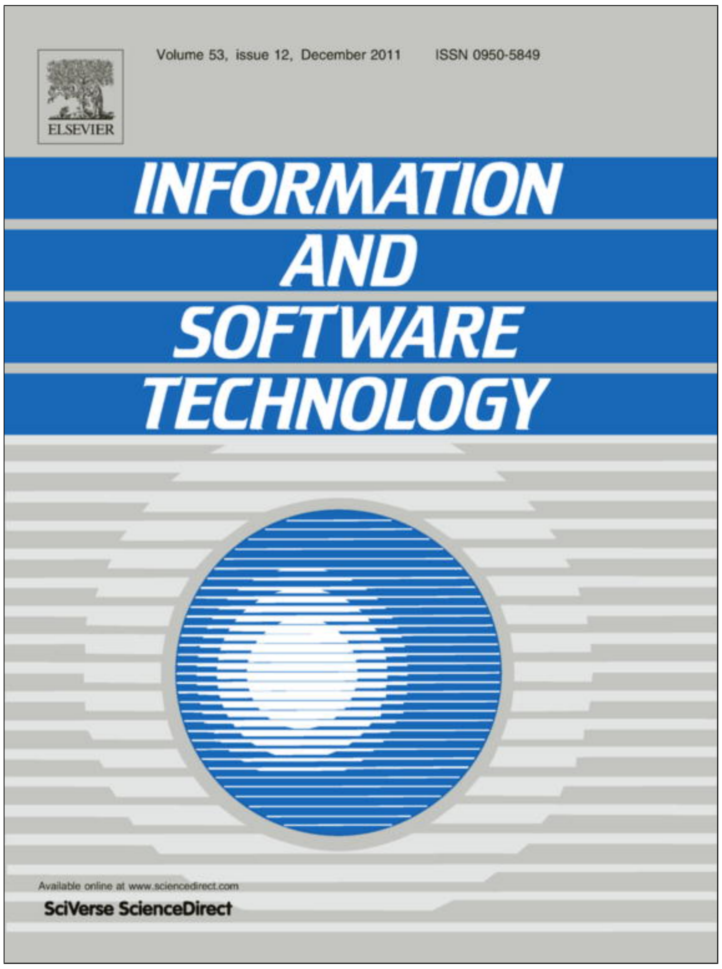
Volume 53, issue 12, December 2011
ISSN 0950-5849
ELSEVIER
INFORMATION AND SOFTWARE TECHNOLOGY
Available online at www.sciencedirect.com
SciVerse ScienceDirect

Information and Software Technology 53 (2011) 1319–1336

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

A comparative study of challenges in integrating Open Source Software and Inner Source Software

Klaas-Jan Stol [a,*], Muhammad Ali Babar [b], Paris Avgeriou [c], Brian Fitzgerald [a]

[a] Lero—The Irish Software Engineering Research Centre, University of Limerick, Ireland
[b] IT University of Copenhagen,Rued Langgaards Vej 7, 2300, Copenhagen, Denmark Denmark
[c] University of Groningen, Department of Mathematics and Computing Science, Nijenborgh 9, 9747 AG, Groningen, The Netherlands

ARTICLE INFO

Article history:
Received 25 January 2011
Received in revised form 13 June 2011
Accepted 14 June 2011
Available online 24 June 2011

Keywords:
Open Source Software
Inner Source
Software development
Challenges
Case study
Empirical studies

ABSTRACT

Context: Several large software-developing organizations have adopted Open Source Software development (OSSD) practices to develop in-house components that are subsequently integrated into products. This phenomenon is also known as "Inner Source". While there have been several reports of successful cases of this phenomenon, little is known about the challenges that practitioners face when integrating software that is developed in such a setting.
Objective: The objective of this study was to shed light on challenges related to building products with components that have been developed within an Inner Source development environment.
Method: Following an initial systematic literature review to generate seed category data constructs, we performed an in-depth exploratory case study in an organization that has a significant track record in the implementation of Inner Source. Data was gathered through semi-structured interviews with participants from a range of divisions across the organization. Interviews were transcribed and analyzed using qualitative data analysis techniques.
Results: We have identified a number of challenges and approaches to address them, and compared the findings to challenges related to development with OSS products reported in the literature. We found that many challenges identified in the case study could be mapped to challenges related to integration of OSS.
Conclusion: The results provide important insights into common challenges of developing with OSS and Inner Source and may help organizations to understand how to improve their software development practices by adopting certain OSSD practices. The findings also identify the areas that need further research.

1. Introduction

Software-developing organizations continuously need to improve their software development processes in order to decrease costs and stay competitive. As a result, methods that have been shown to be successful tend to be imitated in order to reproduce that success within the organization [1]. One such software development approach that has gained much attention over the last decade is Open Source Software (OSS) development. While there is no "standard" set of practices with OSS development (OSSD), some common practices include universal, immediate access to all project artefacts (e.g., source code), release early and often [2], and making local changes to the software (which may lead to a separate "fork").[1] This phenomenon of adopting OSSD practices within a corporate setting is known as Inner Source [4], though other terms are in use as well, such as Corporate Open Source [5] and Progressive Open Source [6]. In this paper, we will use the term "Inner Source". Converting a product's users into its co-developers may improve quality and gain specialized new features that may turn out to be important to a wider audience [7]. Mockus and Herbsleb also suggest that commercial software development can benefit from certain OSS practices [8], while other researchers have suggested that lessons can be learned from OSS communities [9,10]. Several success stories and lessons learned have been reported based on case studies in a number of large organizations that have adopted Inner Source, such as Alcatel-Lucent [5], HP

[1] Forking the development of an OSS project into a new, separate project is typically frowned upon [3], as a fork splits a community into two smaller, competing communities. It is only used as an extreme measure if the splitters have strong disagreement about the direction of the OSS project's development. A recent example is LibreOffice, which is a fork of the OpenOffice.org project started by several members of the OpenOffice.org community.

* Corresponding author. Tel.: +353 61 23 3737; fax: +353 61 21 3036.
E-mail addresses: klaas-jan.stol@lero.ie (K. Stol), malibaba@itu.dk (M. Ali Babar), paris@cs.rug.nl (P. Avgeriou), brian.fitzgerald@lero.ie (B. Fitzgerald).

0950-5849/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2011.06.007

[6], Nokia [11], Philips [4] and SAP [12]. Though OSSD practices are more applicable to large organizations (due to geographic distribution inherent in large organizations), smaller organizations can also benefit from OSS development practices [13]. While each of the abovementioned organizations have adopted a certain set of OSSD practices, it is worth noting that each implementation of Inner Source is uniquely tailored to the organization, which implies that the choice of which practices are adopted, and how, varies per organization. This is further discussed in Section 3.

While these reports of successful adoption of OSS practices are promising, there has been no report on what challenges practitioners can face when *using* an internal "Open Source" project. However, there is quite a large body of research on the challenges of using OSS components in developing products, and in previous research, we identified 21 unique challenges as part of a systematic literature review [14]. This apparent lack of research on the challenges of product development in Inner Source motivated us to undertake a research effort aimed at identifying the challenges being faced by an organization that builds products that include in-house developed software components using OSS development practices (within the organization).

The goal of this research was twofold: first, our intention was to empirically gain an understanding of the adoption of Inner Source and its associated challenges within an organization. Second, we intended to comparatively analyze the challenges of integrating OSS within an industrial environment with the challenges of integrating Inner Source Software. Given the goal of our research, we used the case study research method [15]. This paper reports on design, logistics, and findings from the case study carried out at a large organization, which had adopted Inner Source. The main contributions of this work are to:

- Illustrate one way of implementing Inner Source by identifying adopted OSS practices (addressed in Sections 6 and 7.6).
- Increase our understanding of challenges related to integrating Inner Source Software (addressed in Sections 7.1 to 7.4).
- Compare findings to challenges reported in the literature on product development with OSS components to identify how these manifest in a corporate setting (addressed in Section 7.5).

The results presented in this paper can be helpful to other organizations that have adopted, or are planning to adopt OSS practices when they wish to compare their approach. Furthermore, the identified challenges and approaches can help to form a research agenda for the software engineering research community in general, and the OSS research community in particular.

This paper's structure is based on a study by Petersen and Wohlin presented in [16], in which they present a comparison of issues and advantages in agile and incremental development between the literature and an empirical case study. We present a similar structure: we draw a comparison between challenges in integration of OSS as reported in the literature on the one hand, and challenges related to integration of software developed in an Inner Source environment that were identified through an empirical study on the other hand. We therefore use Petersen and Wohlin's diagrammatic approach to outline the structure of this paper, which is presented in Fig. 1.

In Section 2 we discuss the terminology used in the remainder of this paper. Section 3 presents background information, outlining the context of this study and related work. Section 4 presents the results of a systematic literature review, through which we identified 21 challenges that have been reported in the literature relating to product development using OSS. Section 5 presents the research design of the case study. Section 6 outlines the implementation of Inner Source in the studied organization (which we refer to as "SoftCom"), which gives insight into the context of our study. Section 7 presents the results of the case study, and a comparison to the findings of the literature review. We discuss implications of the results in Section 8. Section 9 concludes this paper and presents an outlook to future work.

2. Terminology

As noted above, there is no commonly accepted term for the concept of adopting OSS development practices within a corporate context. Different organizations have used different terms: at Philips Healthcare, it is called "Inner Source" or "Inner Source Software" [4], while at Alcatel-Lucent it is called "Corporate Open Source" [5]. At Hewlett–Packard Corp. this phenomenon is referred

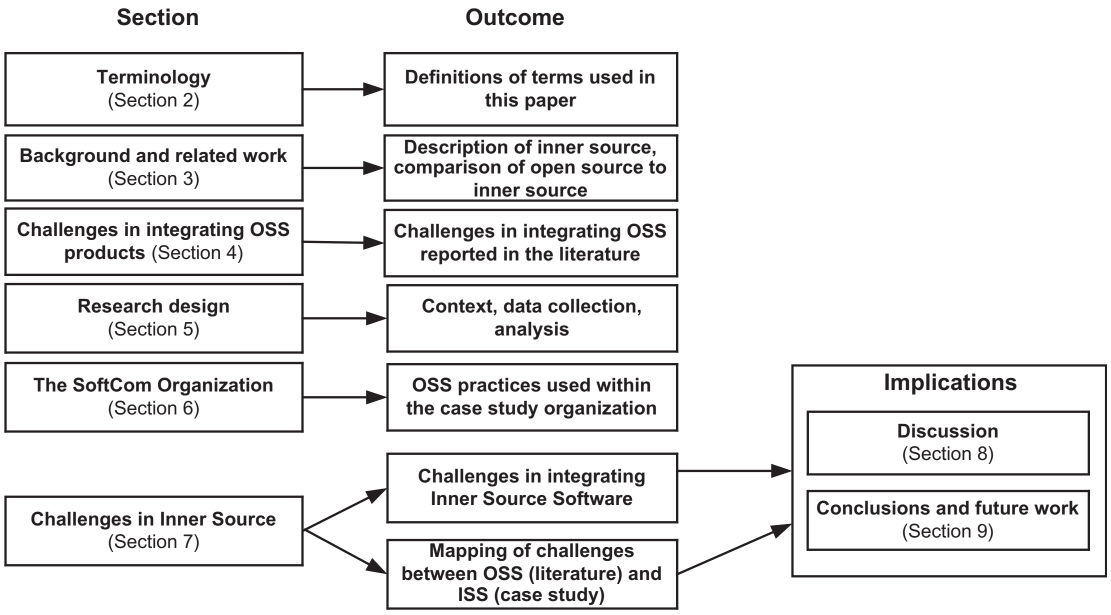
Section
Outcome
Terminology (Section 2)
Definitions of terms used in this paper
Background and related work (Section 3)
Description of inner source, comparison of open source to inner source
Challenges in integrating OSS products (Section 4)
Challenges in integrating OSS reported in the literature
Research design (Section 5)
Context, data collection, analysis
The SoftCom Organization (Section 6)
OSS practices used within the case study organization
Challenges in Inner Source (Section 7)
Challenges in integrating Inner Source Software
Mapping of challenges between OSS (literature) and ISS (case study)
Implications
Discussion (Section 8)
Conclusions and future work (Section 9)

Fig. 1. Structure of this paper.

to with the umbrella term "Progressive Open Source" (POS) [6]. Within POS, three different levels are distinguished: "Inner Source", "Controlled Source" and "Open Source". Within the context of POS, "*Inner Source refers to the application of the Open Source approach and benefits to developers within the corporate environment—i.e., 'open' to all developers behind the firewall*" [6]. The second tier within POS is Controlled Source, which is the same concept, but access is extended to specific corporate partners. The third tier is Open Source as defined by the Open Source Initiative (OSI), and therefore the source code is open to anyone with an Internet connection.

While we do not claim there is no room for different terms, a common vocabulary is desirable for researchers and practitioners to be able to deliberate and discuss research in this area. Furthermore, confusion may arise when using different terms; for instance, the terms "Inner Source" and "Inner Source Software" are very similar, and have been used as synonyms. However, the additional word "software" may tend to imply a product, in the same way that Open Source Software refers to a software product. Both terms have been used to refer to the development practices; that is, a *process*, rather than a *product*. Moreover, "Inner Source" within the context of POS is slightly different from the term Inner Source as used for instance by Wesselius, who defines it as a "*limited environment that has a closed border (such as a company, a division or a consortium)*" [4]. In POS, on the other hand, an environment that limits access to a consortium (i.e., partners) is called "Controlled Source"; in POS there is an explicit distinction with respect to the openness of the source code within one organization and a consortium of organizations (partners).

For these reasons, we provide definitions of a number of terms that we use throughout the rest of this article. We base our definitions on an earlier definition by Gaughan et al. of Inner Source [17] and the common understanding of the terms Open Source, Open Source Software, and Open Source Software development (OSSD). We define the following terms:

2.1. Inner Source

The leveraging of Open Source Software development practices within the confines of a corporate environment. As such, it refers to the *process* of developing software at an organization that has adopted OSS development practices.

Synonyms for this are the terms "Corporate Open Source" and "Progressive Open Source", as used by other authors. This definition subsumes the meaning of "Controlled Source" in POS; that is, Inner Source may occur within the context of a single organization, or a consortium of organizations.

2.2. Inner Source Software (ISS)

The software product that is developed within an Inner Source context. That is, the *product* that is developed at an organization that has adopted OSS development practices.

This definition differs in meaning to that used by other authors, where it would be a synonym for "Inner Source". We note that the source code is still closed, and is therefore different from OSS.

2.3. Inner Source Software Development (ISSD)

The development of a specific software product (namely, Inner Source Software) within an Inner Source environment; that is, an organization that has adopted OSS development practices.

This is similar to the term OSSD, which refers to the development of Open Source Software. Therefore, ISSD refers to a process as well, just as Inner Source, but whereas Inner Source refers to the *phenomenon* of adopting OSSD practices within a corporate environment, ISSD refers to the more concrete notion of development of a specific Inner Source Software product.

3. Background and related work

In this section we review the work related to our study to discuss relevant concepts that will be used for interpreting and discussing the findings from this study.

3.1. Developing with Open Source Software

Organizations may exploit OSS in different ways in the context of software development. Both Van der Linden [18] and Hauge et al. [19] identified five different options:

1. adopting development practices ("Inner Source");
2. using OSS development tools;
3. integrating OSS components;
4. publishing in-house developed components as OSS;
5. establishing a symbiotic relationship with an OSS community.

In this research we focus on the first option; that is, the adoption of OSS *practices*. We note that these five options to adopt OSS are not mutually exclusive. For instance, the first option (and studies reporting on it) often (but not necessarily) includes option two [2] (the adoption of OSSD tools) while the reverse is not true. Furthermore, in Section 3.4 we compare integration of OSS components to integration of Inner Source Software, thereby linking options one and three.

3.2. Open Source Software development practices

OSS is often characterized as software developed by geographically distributed volunteers, lacking work assignments and project planning, and rapid release-and-fix development cycles. However, as Østerlie and Jaccheri argue, there is no empirical evidence that supports the assertion that OSS development (OSSD) is such a homogeneous phenomenon [20]. Feller and Fitzgerald argue that there is no single OSSD process, and note that 'just a handful of projects' (i.e., Mozilla Firefox, Apache, Linux kernel) keep recurring in OSSD research [21]. Similar results were reported in [22]. Each OSS project may follow different practices, but all OSS projects share a common philosophy[2] [23]. A number of common practices that can be found in many projects are listed in [2]. These observations raise the question what it means for an organization to 'adopt OSS practices'. This is reflected by the fact that there is no commonly accepted definition of Inner Source, let alone agreement on the term itself, as discussed in Section 2. However, our literature review has identified a number of recurring aspects in Inner Source: distributed development [18], 'open' development practices (within an organization's boundaries [4,17]) such as peer-review (code inspection), option to contribute by anyone within the organization and availability of the source code [6]. Gaughan et al. [17] attempted to characterize the Inner Source phenomenon by studying seven cases of Inner Source adoption. They conclude that each case was a 'unique' implementation of Inner Source. Their study produced a list of various motivating factors for implementing Inner Source, as well as benefits that have been reported after the fact. These include: code quality, community debugging and faster development.

[2] We acknowledge there are ideological differences between "Free Software" and "Open Source Software", as outlined by Eric Raymond in his seminal work "The Cathedral and the Bazaar". In the context of our research, however, these ideological differences are not relevant.

Table 1
Key differences between infrastructure-based and project-based Inner Source models.

Characteristic	Infrastructure-basel	Project-based
Reuse	Opportunistic, ad hoc. Maximize the number of projects to be shared within the organization	Strategically planned. Optimizing reuse of critical assets
Support	Optional, differs per project, and dependent on owner/maintainer and "community" activity	Essential for success of Inner Source initiative
Owner/maintainer	Individual project creator/owner(s)	Central core team
Type of software packages	Discrete software packages (e.g., utilities, tools, compilers, shells)	Critical assets (e.g., platform of a Software Product Line). Primary technology, rather than tools and utilities

3.3. Inner Source models

Gurbani et al. [24] describe two different models to implement an Inner Source program: *infrastructure-based* and *project-based*. We discuss both models below.

3.3.1. Infrastructure-based Inner Source model

In an infrastructure-based Inner Source model, an organization provides the required infrastructure (e.g., mailing lists, code repositories, tools) to allow developers host individual software development projects. There have been several reports on what can be considered infrastructure-based Inner Source initiatives. Riehle et al. discuss this model in more detail and how it was applied at SAP [12]. Lindman et al. [11] describe the "iSource" initiative at Nokia, which is also an infrastructure-based program. Dinkelacker et al. [6] named the leveraging of OSS methods and tools within HP Corporation "Progressive Open Source" (POS) (see Section 2). POS has been applied within HP Corporation through two programs: (1) Corporate Source initiative (CSI, POS' first tier, which is called "Inner Source"), and (2) Collaborative Development Program (CPD, POS' second tier, which is called "controlled source"). Both CSI and CDP are infrastructure-based programs. The translation process of OSSD into practices within HP (within its POS program) and its partners has been further reported in [1,25].

3.3.2. Project-based Inner Source model

In the project-based Inner Source model, there is one division (called the "core team") within the organization funded by other divisions that take over the responsibility of a critical resource (shared asset) and makes it available to the other divisions as a shared asset [24]. Gurbani et al. have reported on a project-based Inner Source model applied at Alcatel-Lucent [5,24], where the shared asset was a telecommunications signaling server used in a Software Product Line (SPL) [26]. Wesselius reports on a project-based Inner Source model as applied at Philips Healthcare and discusses business model aspects [4]. At Philips Healthcare the shared asset is a platform for product lines in the medical domain.

3.3.3. Comparison of infrastructure-based and project-based Inner Source models

The previous two sub-sections briefly discussed some typical characteristics of the two Inner Source models. It is important to note that these are *typical* characteristics, based on observations from the literature and our case study.[3] Table 1 provides an overview of the key differences between the two Inner Source models. We briefly discuss them below.

3.3.3.1. Reuse. Inner Source facilitates reuse of software, but the way this is done in the two models is different. Reuse in the infrastructure-based model is typically opportunistic and ad hoc, since the main goal is to maximize the number of projects to be shared within the organization. Project-based Inner Source, on the other hand, is typically strategically planned, and its main goal is to optimize the reuse of critical assets within the organization. The project-based Inner Source cases that have been reported in the literature so far, both consider a single shared asset. Therefore, evaluation, selection and adoption of ISS components are relevant in the infrastructure-based Inner Source model, but not so much in the project-based model.

3.3.3.2. Support. As a result of the reuse focus, there is a difference in the level of support that is provided. In the infrastructure-based model, where projects are shared as the project's initiator/creator sees fit, the level of support is dependent on the "community" that the project attracts. Support is therefore optional and very dependent on the project. Some projects may attract a lot of interest, whereas others do not. The project-based model on the other hand requires that there is support for the shared asset, since it is part of the organization's strategy. Without support, business units/projects that use the shared asset may run into too many difficulties. Without sufficient support, the business strategy may be jeopardized.

3.3.3.3. Owner/maintainer. Software projects in the infrastructure-based model are owned and maintained by the individual project's creators/owners. Maintenance is therefore dependent on the maintainer of the project and the community that the project attracts (similar to support), who may be busy with his/her normal development activities. In the project-based approach, there is typically a separate core team, which has been established in the organization as an independently funded group, that has formal ownership of the shared asset.

3.3.3.4. Type of software packages. The types of software packages that are made available in the infrastructure-based model are typically packages such as tools and utilities, including compilers and shells. Such tools or other utilities are shared throughout the company in order to allow others reuse it and save efforts of having to recreate such tools. In the project-based model, on the other hand, the types of software are typically business-critical assets that are essential to the products being developed. Such an asset could be for instance the platform in a Software Product Line (SPL).

3.4. Comparing integration of Open Source Software and Inner Source Software

In Inner Source, software is developed in-house, using practices borrowed from the OSSD philosophy. When building systems using ISS components, these need to be integrated, in a similar way to building systems with OSS components. The difference between these scenarios is that in the one case (Inner Source), the development community is established within the boundaries of the software-development organization, and in the other case (Open Source) the community is outside the organization. However, during the integration of such components, the interaction dynamics

[3] The authors gratefully acknowledge Vijay K. Gurbani for useful advice as to the differences between the two inner source models (personal communication).

between the integrator and supplier (that is, "the community") are comparable.

In this study, we argue that "*Inner Source*" (option 1 in Section 3.1 above) and "*integrating OSS components*" (option 3 in Section 3.1 above) are comparable from the integrator's perspective. In the one case, the software-developing organization has an "internal" OSS project (limited within the boundaries of the organization) that is integrated into a final product, while in the other case the OSS project is external. In both cases, products are built with these components. This is illustrated in Fig. 2 and further explained in the following paragraphs.

The diagram on the left-hand side in Fig. 2 shows two software-developing organizations (A and B) that integrate an OSS product. The OSS is developed by contributors dispersed all over the world, who communicate through the Internet (e.g., through mailing lists, defect trackers and IRC channels). In other words, the OSS community is *external* to the organization. Sometimes, organizations that integrate OSS products can be active members of such a community by contributing (e.g., bug reports and feedback, feature requests and source code) or sponsoring.

The middle diagram in Fig. 2 shows a software-developing organization with two divisions (A and B) that integrate Inner Source Software (ISS). The software is developed within an organization that has adopted Inner Source, and therefore uses certain development practices common in OSSD. In other words, the organization integrates software that is developed using similar practices as used in OSSD, and has effectively an internal "community", or "market place" [4]. Wesselius called this "*creating the bazaar within the cathedral*" [4], using Raymond's metaphors [3]. In both OSSD and ISSD, developers may choose *voluntarily* to contribute to the ISS product (shared asset).

The diagram on the right-hand side in Fig. 2 is included to emphasize the difference with a "traditional" (non-inner source) software development organization. We acknowledge that there is no single type of "traditional" software organization. However, it has become common practice to integrate Commercial Off-The-Shelf (COTS) components in final products [27]. The focus of all the diagrams in Fig. 2 is on the integration of components, and to illustrate by whom these are developed. Furthermore, the issue of ownership is highlighted here: OSS products are "owned" by the community (protected by an OSS license, such as the GNU General Public License (GPL)). ISS components, on the other hand are still closed-source (but "Open Source" within the organization's boundaries). COTS components are typically closed-source, and are owned by the third-party component supplier.

Integrating OSS and ISS components is similar, because in both cases integration is done in a similar fashion. For instance, the integrator has direct access to the component (as opposed to ordering a commercial component, which may be more time-consuming), and has the option to customize the component to the specific needs of the business division. Furthermore, the integrator may be faced with similar challenges, for instance, challenges in handling extensions and modifications [28]. We therefore argue that it is informative to compare these two cases, as research on OSSD is much more mature than on ISSD and important lessons can be transferred from the former to the latter.

4. Challenges in integrating Open Source Software products

In [14] Stol and Ali Babar present a review of the literature of challenges in using Open Source Software in product development. Papers were partly identified through a search of the literature following rigorous guidelines for conducting a systematic literature review in [29], to ensure that as many relevant studies as possible were included.

Table 2 lists the challenges that have been identified in the review of the literature. We use the challenges' identifiers as in [14] (C1 to C21), which are used throughout discussion of the results in this paper. The identified challenges have been classified into six different categories. Three challenges are related to **product selection**, two challenges were reported relating to **documentation**. Six challenges were classified in the category "Community, support and maintenance". These are connected to the relation of the organization using the OSS product and the OSS product's community. Since maintenance is closely related to support (and usually provided by the community), these challenges are closely related and therefore classified into one category. Five challenges

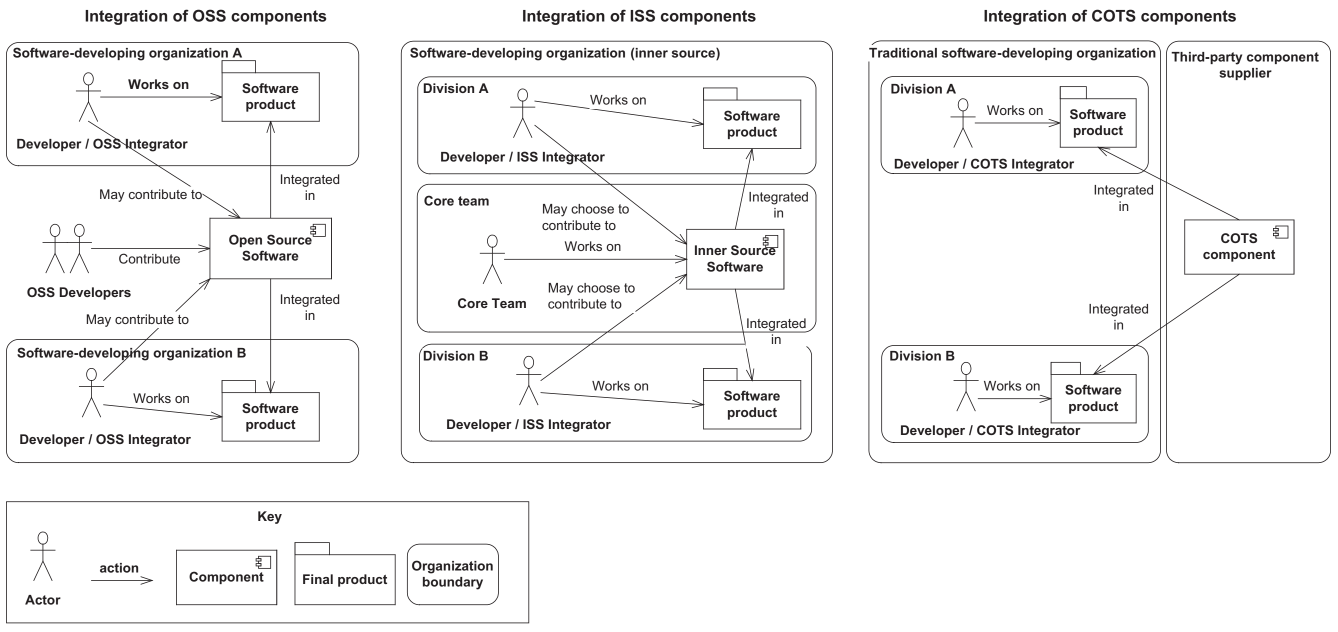
Integration of OSS components
Software-developing organization A
Works on
Software product
Developer / OSS Integrator
May contribute to
Integrated in
Open Source Software
Contribute
OSS Developers
May contribute to
Integrated in
Software-developing organization B
Works on
Software product
Developer / OSS Integrator
Integration of ISS components
Software-developing organization (inner source)
Division A
Works on
Software product
Developer / ISS Integrator
Core team
May choose to contribute to
Works on
Integrated in
Inner Source Software
Core Team
May choose to contribute to
Integrated in
Division B
Works on
Software product
Developer / ISS Integrator
Integration of COTS components
Traditional software-developing organization
Third-party component supplier
Division A
Works on
Software product
Developer / COTS Integrator
Integrated in
COTS component
Integrated in
Division B
Works on
Software product
Developer / COTS Integrator
Key
Actor
action
Component
Final product
Organization boundary

Fig. 2. Integration of OSS components (left), ISS components (middle), and COTS components (right). OSS components are developed and "owned" by its community, whereas ISS components are developed and owned by an organization. COTS components are purchased from third-party suppliers.

Table 2
Challenges in integrating OSS in product development.

Category	ID	Challenge	Reported in
Product selection	C1	Identifying quality products among the large supply is difficult due to uncertainty about quality (e.g. usability, stability, reliability)	[30–36]
	C2	Lack of time to evaluate components	[31]
	C3	Decide what "fork" of the project should be chosen	[37]
Documentation	C4	Lack of, or low quality documentation	[31,32,38,39]
	C5	Several descriptions of the same component	[40]
Community, support and maintenance	C6	Dependency on the community for further support and upgrades; possible need to hire additional talent for maintenance; difficult to control the quality of the support; lack of helpdesk and technical support	[32–36]
	C7	Custom changes need to be maintained, which is time-consuming and may cause problems with future versions/community may take a different, incompatible approach	[28,30,37,41–43]
	C8	Convincing OSS community to accept changes (modifications may be too specific); contributions can be difficult or costly. Difficult to control the architecture if not a core member	[28,30,32,37,43]
	C9	Uncertainty about product future and consequences for company product	[37]
	C10	Community members would like to have a bigger say in features and integrating final product with company	[41]
	C11	Contributing and investing in OSS project costs resources	[41]
Integration and Architecture	C12	Backward compatibility concerns	[36,41]
	C13	Modifications needed to implement missing functionality or fit into architecture	[28,36]
	C14	Incompatibility between components or existing systems	[34,36]
	C15	Horizontal integration	[32]
	C16	Vertical integration/mismatch of platform/programming language	[32]
Migration and usage	C17	Complexity of configuration	[36]
	C18	User training/learning costs	[34,36]
Legal and Business	C19	Complex licensing situation	[36,39,41,44,45]
	C20	Concerns about, or no clear strategy on Intellectual Property & Rights issues	[36,44,45]
	C21	Lack of clear business models that are appealing to industry	[34,45]

are classified into the category "Integration and Architecture". These are challenges related to integration of products and are typically related to the product's architecture. Two challenges were related to **migration and usage**. These challenges are encountered as a result of migrating to (replacing other products with) OSS products, and using and configuring products. The last three challenges are classified in the category "Legal and Business" and are related to licensing issues and business models. The challenges listed in Table 2 are used in the comparative analysis, discussed in Section 7, of the findings from the case study reported in this paper.

Studies reporting case studies of Inner Source so far focus on experienced challenges and outline "lessons learned". Most notable are the studies of Inner Source at HP [1,6,25], Alcatel-Lucent [5,24], and Philips Healthcare [4,18]. These studies all report on how OSS principles have been applied within these organizations, and focus on what works and what does not work within a corporate environment. However, none of these studies take an explicit integrator's point of view: what are concrete challenges of integrating software developed using OSS principles?

In order to shed light on this, we decided to conduct an empirical study. Furthermore, we argue that integrating OSS and ISS is similar in certain aspects, and make a comparison with challenges that have been reported in the literature.

5. Research design

This study aims at increasing our understanding of Inner Source adoption and challenges within an Inner Source Software development setting. We assert that each implementation of Inner Source is tailored to a particular organization; hence, it is imperative to first understand what OSS development practices an organization has adopted. We address this in Section 6.

After outlining the Inner Source development practices in this case study, we were interested in identifying the challenges that arise when integrating software components developed in-house through applying OSSD practices. Hence, our first research question is (addressed in Sections 7.1–7.4):

RQ1: What are the challenges in developing and using software that is developed as a shared asset?

We then intended to compare these results to the findings of the literature review that had identified challenges in using OSS components in product development. Hence, our second research question is (addressed in Section 7.5):

RQ2: What are the similarities between challenges in integrating OSS and challenges in integrating ISS?

We were also interested in identifying the approaches that the studied organization had adopted to address these challenges. Therefore, our third research question is (addressed in Section 7.6):

RQ3: What are the approaches used to address challenges related to integrating a shared asset?

5.1. Research method

Edmondson and McManus suggest that the research design should be based on the state of prior research and theory [46]. Research on Inner Source has been limited and is still in its nascent phase with little theory available that explores the challenges that organizations may face. Hence, we used case study as our research strategy. Case study approach is considered suitable to investigate a contemporary phenomenon within its real-life context, especially when the boundaries between phenomenon and context are not clearly evident [15]. In this research, case study approach is justified since the implementation of Inner Source is tailored to the specific characteristics of an organization [17]. As Verner et al. [47] point out, case studies may be descriptive, explanatory, exploratory or evaluatory. Given the nascent phase of research in Inner Source, we conducted an *exploratory* case study.

The unit of analysis in this case study is an organization that has adopted an Inner Source approach as a whole. We conducted this study at one of the locations of a large (globally distributed) organization, which has been involved in several OSS related projects and has adopted a *project-based* Inner Source program. The organization was approached through our professional contact. The organization develops both hardware and software for safety–critical systems. For confidentiality reasons, we cannot report the studied organization's specific domain. In the remainder of this paper, we will refer to the organization by the name "SoftCom".

5.2. Data collection

We collected data through eleven in-depth face-to-face interviews. Table 3 lists the participants, their division in SoftCom, and their work experience in years. We refer to the participants by numbers P1 to P11 in order to protect their privacy. Participants had various positions in different divisions within SoftCom, such as business division manager, product manager, technology officer, software architect, software designer and product coordinator, providing us with a rounded perspective from different points of view. A number of them were members of the core team that is responsible for developing the shared asset (see Section 3.3). Most managers also had prior technical experience as a software architect. All participants had extensive experience and knowledge about SoftCom as they had worked there from 10 to more than 25 years.

Prior to conducting the case study, we developed an interview guide [48]. We chose to conduct semi-structured interviews, as these are expected to give a researcher the flexibility to go deeper into unforeseen types of information that can emerge during interviews [49]. All interviews were conducted at SoftCom's location by the first author. Our contact at SoftCom made local arrangements and scheduled the interviews. After receiving the contact information of all scheduled participants, we sent them an introductory letter in which we outlined the aim and procedure of the research. All interviews lasted between 40 and 60 min, and were digitally recorded with the participants' consent. The recordings were transcribed verbatim, in order to record as many details as possible. This resulted in approximately 150 (A4 size) pages of text.

Finally, all recordings were played back once more and cross-checked with the transcriptions in order to make sure that no information was lost during the transcription.

5.3. Data analysis

Data analysis is an iterative process, in particular when the researcher is confronted with a large amount of data (in our case 150 pages of transcripts). Though the research questions are clearly defined, in order to be able to manage the large amount of data collected, we decided it was important to reconstruct the "story line" for each participant, and identify common themes and topics in order to be able to compare these topics. Since different participants sometimes used different descriptions for their experiences and insights, it is important to identify these common themes. This is a form of triangulation (across data sources, namely the participants of our study), which is a common procedure to establish validity in qualitative studies [54].

We analyzed the data as follows. All interview transcripts were thoroughly read, and phrases of interest were coded with labels to reflect the topic of that phrase, following the approach described by Seaman in [49]. The coding was performed by the researcher who had conducted the interviews. In order to ensure reliability of our findings, we applied another form of triangulation, namely among different investigators [54]; two researchers have discussed the findings in several face-to-face meetings.

Using specialized software for qualitative data analysis (NVivo), we constructed a small set of preformed labels referring to topics that we expected to arise from the data, and which were also of interest to us. During data analysis, this set of labels evolved; labels were merged, added and deleted. After the initial coding, we looked at groups of coded phrases and merged them into categories. This structuring of the data helped us to understand and manage the large amount of information. Per category, the labeled text was exported to a Microsoft Word document, thereby grouping all related paragraphs on a particular topic in one document. This allowed us to further read and analyze the data per topic.

After we had acquired an initial overview and understanding of the data, the first author created memos in the form of visualizations of the transcripts. These were simple box-and-line diagrams; part of such a memo is shown in Fig. 3.

Boxes represent the main topics, whereas each box may have a number of "attributes", or sub-topics, which are short phrases connected by lines. Boxes can also be related to other boxes. The first author created such visualizing memos for each interview, which were "maps" of the transcripts' contents, and could quickly communicate the contents of the interviews to the other three researchers.

After identifying the main topics of each interview and recognizing common themes among the different interviews, the first author re-read the coded transcripts, to identify the challenges and approaches that participants had reported. In some cases it was necessary to refer back to the original transcripts to refresh the researcher's mind of the context. The participants often mentioned an approach to address a certain challenge after reporting that challenge. Sometimes this was clearly indicated by key phrases, such as: "*So what we do now, is [...]*" or "*In order to address this [...]*". The challenges and approaches were listed in a spreadsheet, together with source information to easily back trace the original phrasing in the transcripts. Similar entries were merged.

We had noticed that many challenges reported by different participants were related (causing or exacerbating other challenges). In order to explore and visualize these relationships, we drew more box-and-line diagrams (separate from those shown in Fig. 3). In these diagrams, boxes represented challenges and approaches, and lines represented relationships (such as "addresses", "exacerbates", etc.). These visualizations finally evolved into a complete diagram, shown in Fig. 3. This procedure of establishing a traceable, documented justification of the analysis process (transcription of the interviews, coding, memoing) by which conclusions are reached is called an *audit trail*. This is a recommended practice to establish validity in qualitative studies [55].

Each challenge was identified by at least one participant, while 10 (out of 13) were identified by two or more participants. We did not use results from earlier interviews in interviews conducted later; this means we did not let participants comment or confirm on reports from other participants. Therefore, we argue it is not appropriate to perform a frequency analysis of the challenges and approaches, as this may misrepresent the truth. Each

Table 3
Participants in our study.

ID	Division	Experience (years)
P1	Division A	20
P2	Core team	10
P3	Division B	15
P4	Core team	17
P5	Technology office	26
P6	Core team	25
P7	Division C	10
P8	Core team	10
P9	Technology office	25
P10	Core team	13
P11	Division D	12

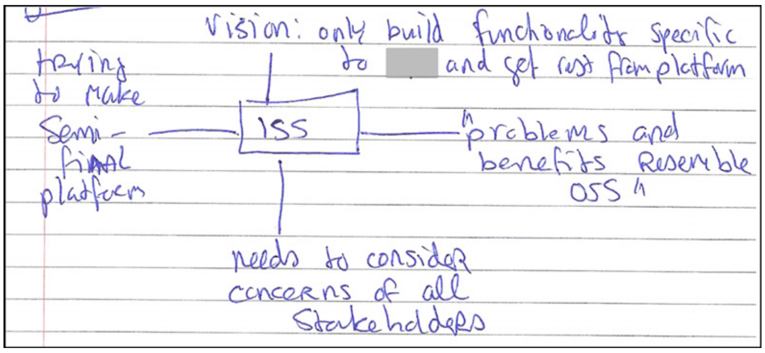
vision: only build functionality specific to [redacted] and get rest from platform
trying to make semi-final platform
ISS
"problems and benefits Resemble OSS"
needs to consider concerns of all Stakeholders

Fig. 3. Example of "box-and-line" visualization.

participant told his/her story (following our questions from the interview guide), highlighting his/her view on the studied topic.

6. The SoftCom organization

In this section, we describe details of SoftCom that are relevant to this study. We also report the key OSS practices that have been adopted as part of the Inner Source initiative in SoftCom. This sets the context in which the findings reported in Section 7 should be interpreted as each implementation of Inner Source is tailored to the organization's specific context and needs (as we asserted in Section 3.2).

SoftCom's products are developed as members of a SPL. Initially, a common platform used by all products in the SPL was provided to business divisions as a binary deliverable. Besides the general motive to increase software reuse, another reason for providing a common platform was that company management had planned a series of company acquisitions, whose products were to be adopted and integrated into a common architecture. Prior experience suggested that by providing a common platform, a *turf battle* about whose technology to use in such acquisition could be avoided. Over the last decade, SoftCom has acquired a number of companies, which have become new business divisions. The software product that a new business division brought in, would be thoroughly scrutinized to see how it would fit with the common platform, and what parts could be adopted in the platform. On the other hand, the new business division would have access to the platform that provides common functionality, and could replace parts of their software by functionality provided by the platform. This results in less code for the business division to test and maintain.

A number of years ago, SoftCom decided to adopt a *project-based* (see Section 3.3) Inner Source approach, in which the common platform is managed as a shared asset. In the remainder of this section, we outline how Inner Source has been adopted in SoftCom. Fig. 4 shows the conceptual model of the adopted Inner Source model. This model closely resembles an OSSD approach reflecting common mechanisms found in OSSD: the Core Team represents the OSS community's core developers; the business division is the OSS integrator that uses the OSS product in product development, and the shared asset is the OSS product. Other parts of Fig. 4 are discussed below.

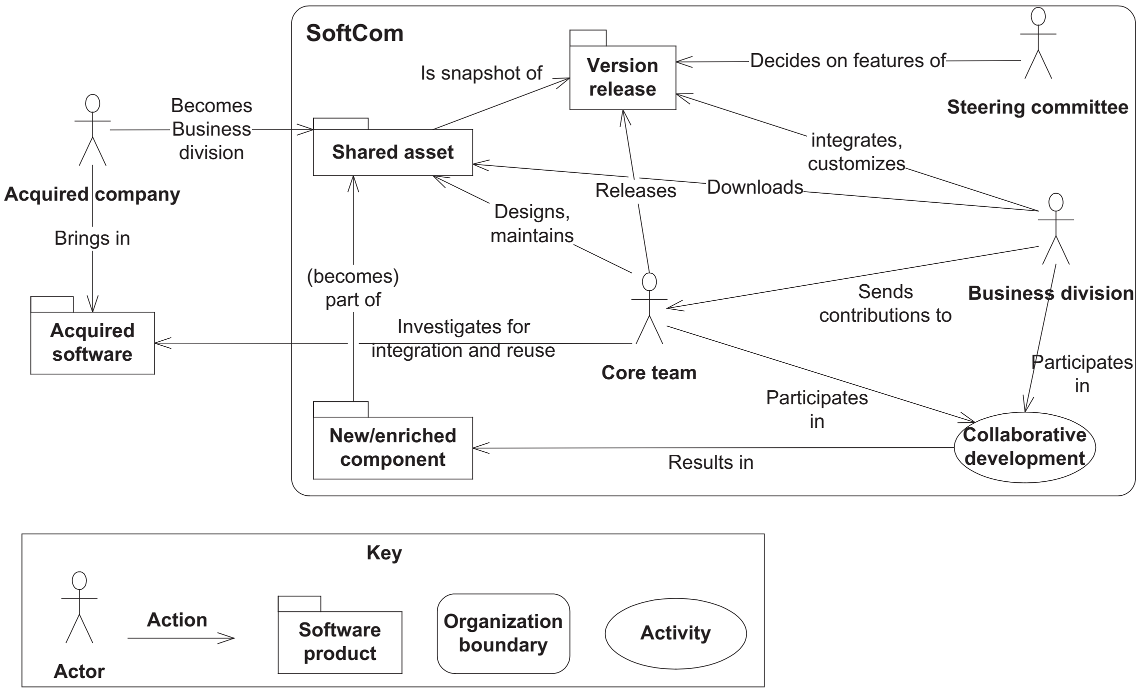

Fig. 4. Conceptual model of Inner Source in the SoftCom organization. Arrows between actors, products or processes indicate the order of reading, e.g., a Business division Participates in Collaborative development.

6.1. Adopted Open Source Software development practices

As previously mentioned, Inner Source refers to leveraging OSS practices within a corporate environment [17]. This does not mean that all OSS practices are suitable to be applied within a corporate setting. Rather, when an organization is involved in commercial software development, it only adopts practices that can help improve process and product quality in such a way that the organization can control the process and the product release deadlines. The level to which an organization is 'going open' differs from one organization to another. Each organization will implement Inner Source in a different way, tailored to the constraints and needs of the organization [17]. This variety of practices is similar to the variety of practices in OSS projects mentioned earlier in this paper (see Section 3.2). In the remainder of this section, we discuss the OSS development practices that have been applied within SoftCom.

6.1.1. Regular releases and frequent integration

As is common in many OSS projects, SoftCom has a *core team*, which makes regular 'stable' releases of the shared asset [2,21]. A *steering committee* consisting of a number of architects decides what new features will be included in the new version. Business divisions can integrate these releases into their product, but they may also choose to follow development of the shared asset more closely by regularly downloading the latest version; the Inner Source model enables this option. By staying closer to the latest version of the shared asset, a business division can reduce its integration efforts, as it no longer needs to make major revisions when switching from one version to another.

6.1.2. Collaborative development

One of the key characteristics of OSS development is that anybody is free to contribute. In OSS development, this typically happens by sending a patch file that contains the changes made to the source code. The patch is then peer-reviewed by trusted contributors that have write ("commit") access to the source code repository. Contributors that have a record of submitting high quality patches may be granted write access. In such a case, the peer-review is effectively post-commit. In a corporate setting, contributions must be more restricted in order to control the quality of contributions, especially in business- or safety-critical systems. The organization has adopted a mechanism called 'collaborative development', in which the core team and a business unit closely collaborate on the development of a new component, or on enriching an existing component. This mechanism helps in making sure that, on the one hand, the component will fit into the architecture of the shared asset, and on the other hand implements the required functionality, as required by the business division's domain experts.

6.1.3. Local changes to the source code

Business divisions are free to make local changes to the shared asset on which they build their product. This may be a solution if a division finds out about missing functionality shortly before a product release, and the core team may be unable to make the required changes on time. This situation can be beneficial to both the business division and the core team since any such changes are 'bought back' by the core team. This way, a business will no longer have to reapply (and maintain) patches whenever a new version of the shared asset is released. The core team may benefit from the domain expertise that the changes may incorporate.

6.1.4. Tool support

While not exclusive to OSS style development, several tools typically used in OSS projects are also used within SoftCom. Development environments are standardized in SoftCom, and managed and supported by support engineers, who are members of the core team. By standardizing the development environment for all business divisions (and the core team), it is easier to ensure that a code check-in does not 'break the build'. In order to address knowledge sharing issues, a wiki was set up through which developers and architects (both from the core team and business divisions) can share knowledge. Though most information comes from the core team, a wiki allows anyone to contribute, which is highly encouraged by the core team. The adopted wiki implementation allows for semantic annotation of the knowledge, which allows it to be reused in different contexts. Besides a wiki, a mailing list was set up that can be used by developers to ask specific questions. Architects regularly look through these questions and answer if they can. An issue tracker is also available to report and communicate problems.

7. Challenges in Inner Source

We have identified 13 challenges in the case study (numbered S1 to S13), listed in Table 4. We classified these challenges using

Table 4
Challenges identified in the case study and references to challenges related to the use of OSS products as identified in the literature (if applicable).

Category	ID	Challenge	Challenge in using OSS
Documentation and knowledge	S1	Lack of documentation	C4
	S2	Core team that develops shared asset lack domain knowledge causing lack of attention for non-functional requirements	n/a
Community, support and maintenance	S3	Core team must balance spending resources over required architectural refactoring and implementing requirements as requested by business divisions	C6
	S4	Business divisions' contributions do not fit	C7
	S5	Core team's reluctance to adopt business divisions' contributions	C8
	S6	Business divisions' reluctance to contribute to the shared asset	C11
	S7	Business divisions treating core team as a traditional component supplier; business division does not have influence on architecture and interfaces	n/a
Integration and architecture	S8	Missing interfaces causing usage of private interfaces, resulting in high integration efforts when switching to a new version	C12
	S9	Missing functionality	C13
	S10	Integrating acquired software into the shared asset hindered by architectural mismatch	C14
	S11	Component-suite model of shared asset allows for too much freedom in usage, causing many test and integration efforts	C15
	S12	Components are not designed for other use-cases (not sufficiently generic)	n/a
Migration and usage	S13	Difficulty in using the shared asset, configuring is complex	C17

the same categories identified in the literature review [14] (the category "documentation" was renamed "documentation and knowledge", since documentation is a means to share knowledge). As noted before, there are no issues in the categories "product selection" and "legal and business".

One of the objectives of our study was to compare the challenges related to ISS to the challenges related to OSS (as identified in the literature). We first present the challenges identified in our case study in Sections 7.1–7.4 (RQ1). We then discuss the mapping and comparison of challenges related to the use of OSS products (from the literature review) separately in Section 7.5 (RQ2). Approaches adopted in SoftCom to address the challenges are presented in Section 7.6 (RQ3).

7.1. Documentation and knowledge

7.1.1. Lack of documentation

A number of participants indicated a lack of knowledge sharing and documentation to be a challenge (S1). Though the development process followed by SoftCom prescribes that design and test packs of documentation are written, one participant stated:

"*Nobody reads those test packs or even the requirements packs.*" —*P10, core team.*

Participants indicated a strong preference of having "How-to" knowledge, and basic design documentation that is needed to use the software in a useful way. In particular, information about interfaces, architectural patterns and tactics were considered to be useful. The lack of knowledge of how to use the software makes using the shared asset difficult, an often-heard challenge in this study. It was felt that, as an integrator, one needs to know too many details about the internals.

7.1.2. Lack of domain knowledge

The core team designs, develops and maintains the shared asset, which is used by the business divisions. However, a challenge that the core team deals with is that they lack specialized domain knowledge of the various business divisions' products (S2). As a result, participants reported a lack of attention paid to the non-functional requirements:

"*If a component does what it needs to do with respect to the functionality, then [the core team] thinks they're done. Non-functional requirements in particular, in the context of using a product, is an obstacle. Performance, resource usage, those are often not considered.*" —*P5, technology office.*

"*The issue is often with the non-functional requirements. It could be that the architecture chosen by the supplier performs badly with our type of data.*" —*P11, business division.*

7.2. Community, support and maintenance

7.2.1. Balancing refactoring and requirements

The core team makes regular releases of the shared asset. As the shared asset evolves, there is a need for refactoring the architecture and making other improvements. However, since business divisions plan their releases based on a new version and require new features, the need of spending resources on these maintenance activities creates difficulties for the core team (S3). One interviewee reported this difficulty as follows:

"*If push comes to shove, and the next release is scheduled, and a part of the budget is reserved for improvements, then customers say: 'nice that you want to do that, but we need feature X or Z, otherwise we can't deliver our product'.*" —*P10, core team.*

7.2.2. Contributions do not fit

The Inner Source model enables and encourages others within SoftCom to make contributions to the shared asset. However, it was found that until some years ago, contributions would not fit the architecture of the shared asset (S4). As one participant reported:

"*People would make additional pieces of software without consultation. And when you try to incorporate that into the platform [...] it turned out to be useless.*" —*P2, core team.*

7.2.3. Reluctance to accept contributions

The core team is responsible for the design and maintenance of the shared asset. The core team may be somewhat reluctant to adopt contributions of business divisions (S5), since this implies adoption of the maintenance responsibility for the contributed software as well. One participant phrased this as follows:

"*I think that if [the core team] would integrate something back into their platform, then from the other groups' point of view they would also assume the responsibility of maintenance. And from that point on they are responsible for those parts. It's not well defined, that if a group gives something back to [the core team], who is responsible for the maintenance of that part? Everybody thinks it would be [the core team]. And that restricts that road back.*" —*P5, technology office.*

Several factors may exacerbate this. Firstly, it may be due to the 'not invented here' syndrome. Secondly, contributions made by business divisions may be too specific for the business division that wrote them, rendering them unusable for other divisions.

7.2.4. Reluctance to contribute

Business divisions are typically not very eager to contribute to the shared asset (S6). One participant explained this situation as follows:

"*Then the issue of maintenance arises: we wouldn't mind publishing [the software], but only if the central group wants to do the maintenance.*" —*P1, business division.*

Another reason for this reluctance appeared to be that a business division considers development of certain type of software to be the core team's responsibility:

"*When we're doing something that's generic, then we try to have it made by [the core team]. [...] Then we don't want to make [that software] ourselves.*" —*P3, business division.*

7.2.5. Core team as traditional supplier

A number of business divisions still treat the core team as a traditional component supplier (S7). Rather than adopting the Inner Source philosophy, these divisions have a more traditional view of software development, and do not benefit from the Inner Source model. One participant described this state of mind in these words:

"*If [the shared asset doesn't provide sufficient functionality], we send the requirements to [the core team]. They will start working on it, if all goes well (laughs). [...] They start working, developing and they're in the basement for a while and then they come up and go: 'Tadaa! Here you go,' and in practice we're not really ready for integration, so we thank them and tell them we'll come back to them. And after a few months you use it, and then all sorts of integration issues arise. And the supplier is already in maintenance mode, and is making new things for other customers, and they don't really have the resources to fix those problems. That's really the biggest issue we have in practice.*" —*P11, business division.*

7.3. Integration and architecture

7.3.1. Changing interfaces

One issue while integrating the shared asset in a product is changing interfaces (S8). Interfaces of components in the shared asset are not well specified and documented, or may be private. One participant said:

> "*In the past there was this whole range of private interfaces that you had to use otherwise you wouldn't get it to work.*" *—P3, business division.*

Our study revealed that other business divisions were also experiencing this challenge as another interviewee reported:

> "*[. . .] So we just use [these private interfaces]. [. . .] Those private things can change and that will happen, and then everything collapses.*" *—P11, business division.*

7.3.2. Missing functionality

A common challenge reported is that functionality is missing in newly delivered components (S9). One participant explained:

> "*When we get the component, it's never been integrated, and when we do that, you find that things are missing, and without that we can't deliver. A car with three wheels is no good. . .*" *—P11, business division.*

7.3.3. Architectural mismatch

As new companies are acquired and integrated into the organization as business divisions, the core team will investigate which parts of the acquired software can be adopted in the shared asset. Incorporating software from new divisions may be troublesome (S10), as one participant described:

> "*We're building a big box of Lego bricks, and they all have the same interface; Lego on Lego fits perfectly. But our stakeholders have an architecture based on Meccano, or something else, and then Lego won't fit, and then you need to write connectors, we call that glue code. And as it turns out, the problems are always in the glue code.*" *—P6, core team.*

7.3.4. Boundless reuse

Initially, the shared asset was organized as a 'component suite', a collection of components. Divisions could take whatever components they needed. However, some (acquired) divisions' systems had only been partly adapted to the shared asset's architecture to allow them to reuse certain components, since the component suite model allows for a 'take what you need' approach. This resulted in significant integration and test efforts that the organization had hoped they could reduce through software reuse, one of the reasons to set up the shared asset in the first place. Furthermore, if the architectures of the application and the shared asset are not well aligned, there may be a need to write connectors (or 'glue code'). Glue code may introduce problems, as described above.

7.3.5. Use-case mismatch

The shared asset contains functionality that is common to all business divisions. New components are being added over time, as new requirements emerge, and new business divisions are acquired by SoftCom. A challenge is to make components generically suitable to all business divisions. A common problem is that components have a use-case mismatch (S12). As one member of the core team explained:

> "*People complain about the maturity of the component. We build them a first time, and they're used by customers X and Y in certain products. [. . .] After a year there's another customer that also wants to use it, but in a slightly different way. Sometimes they think they need to use it differently while that's not the case. They will consider the component as immature, because it doesn't do exactly what they want, or it wasn't tested in that particular use case.*" *—P10, core team.*

A member of a business division phrased it very similarly:

> "*The nature [of integration problems] is usually a slightly different use-case of the component than what [the core team] had tested it for.*" *—P11, business division.*

7.4. Migration and usage

Several participants indicated that it is difficult to use the shared asset (S13). Since the shared asset is the platform for a Software Product Line, it must provide functionality that is usable by different business divisions in different specialty domains of a common industrial domain. The core team is well aware of this issue, which was reported by one of the interviewees in the following words:

> "*We created a platform that is used by all business divisions, and because you need to keep everybody happy, it has many configuration options. [. . .] And I think that's one of the reasons that it's difficult to configure it correctly.*" *—P8, core team.*

Business divisions have difficulty understanding how to use the shared asset, and how it relates to their product:

> "*How do we 'click in' our specific application? That's an interface issue, and well, how to pass in the data.*" *—P11, business division.*

Interestingly, after seeing the situation from another business division's perspective, people would increase their insights. A member of the core team described:

> "*And what we saw was that [at first] many people didn't understand the abstractions and why we needed them. [. . .] And later people [after they had moved to a different department] would come to us and say: 'now I understand why it was designed like this'.*" *—P6, core team.*

7.5. Comparison of challenges with OSS and ISS

We observed that ten out of 13 challenges identified in the case study are similar to the challenges when using OSS products, as identified in the literature review [14].

Table 5
Overview of relevance of challenges to OSS, infrastructure-based Inner Source and project-based Inner Source.

Category (number of challenges)	Relevant to OSS	Relevant to infrastructure-based Inner Source	Relevant to project-based Inner Source
Product selection (3)	Yes	Yes	No
Documentation (2)	Yes	Yes	Yes
Community, support and maintenance (6)	Yes	Yes	Yes
Integration and architecture (5)	Yes	Yes	Yes
Migration and usage (2)	Yes	Yes	Yes
Legal and business (3)	Yes	No	No
Total number of relevant challenges	21	18	15

Not all challenges listed in Table 2 are relevant to project-based Inner Source, or Inner Source at all. Section 3.3.3 has discussed a number of typical differences between the infrastructure-based and project-based Inner Source models. Table 5 provides an overview of the relevance of the different categories to OSS, infrastructure-based Inner Source and project-based Inner Source. The last row indicates the total number of challenges that are relevant to the various cases. Challenges in the category "Product Selection" do not apply to project-based Inner Source, since there is usually only a single shared asset that is developed. Selection is not an issue, since the use of the shared asset is strategic and planned. In infrastructure-based Inner Source, where departments or individuals make freely available software on an internal repository, these challenges *could* occur.

Challenges in the category "Legal and Business" are not relevant to Inner Source, since the software is closed source (which rules out any OSS license-related issues) and Intellectual Property & Rights (IP&R) and business-related concerns would not occur.

Fig. 5 shows a mapping of the challenges identified in this case study (middle layer) to the challenges identified in the literature (top layer) indicated; the figure shows only the 10 challenges (out of 15 relevant to project-based Inner Source, see Table 5) identified in the literature that have also been identified in the case study (listed in Table 4). The bottom layer in the figure shows approaches that the studied organization has taken to address some of the challenges.

Below we discuss how the challenges related to integrating OSS identified in the literature manifest themselves in SoftCom; this mapping is indicated by open (white) arrows between elements of the middle layer to elements of the top layer of Fig. 5.

During analysis we found that certain challenges cause or exacerbate other challenges. These "root" challenges are displayed at the bottom in the middle layer whereas the challenges they cause (or exacerbate) are shown at the top of the middle layer. Pointy arrows between elements in the middle layer express a "cause" relationship, e.g., a lack of documentation may cause (or exacerbate) that contributions do not fit. Note that a challenge could be neither a "root" challenge nor caused by another (e.g., challenge S3). Furthermore, we cannot claim that the identified "root" challenges are the *only* sources that cause certain other challenges; it is possible that other factors are at play that have not been identified in this study.

Closed (black) arrows between elements in the bottom layer (approaches) and the middle layer indicate an "address" relationship, e.g., "providing training" addresses the challenge "difficulty in using shared asset". There are three challenges that could not be mapped to any of the challenges identified in the literature (S2, S7, S12); these are colored light gray. Three challenges are not addressed by any approach (S3, S5, S6); these are coloured dark gray in Fig. 5. We discuss the mapping of challenges in the remainder of this subsection. (We only discuss those challenges that were also identified in the case study.)

7.5.1. Lack of documentation (C4)

Lack of documentation is a common complaint in OSS, and in software in general. It was no surprise to us that in this study a lack of documentation was raised as a challenge. A lack of documentation (on how to use the particular product) was considered to be an issue, and it also exacerbated the challenge of using the shared asset.

7.5.2. Dependency on community (C6)

Business divisions that base their product development on the shared asset have a dependency on the core team in a similar way to an organization using an OSS component that becomes dependent on the community for new updates. In Inner Source, the core team must balance its resources spent on providing implementation of new features on the one hand, and performing architectural refactoring on the other.

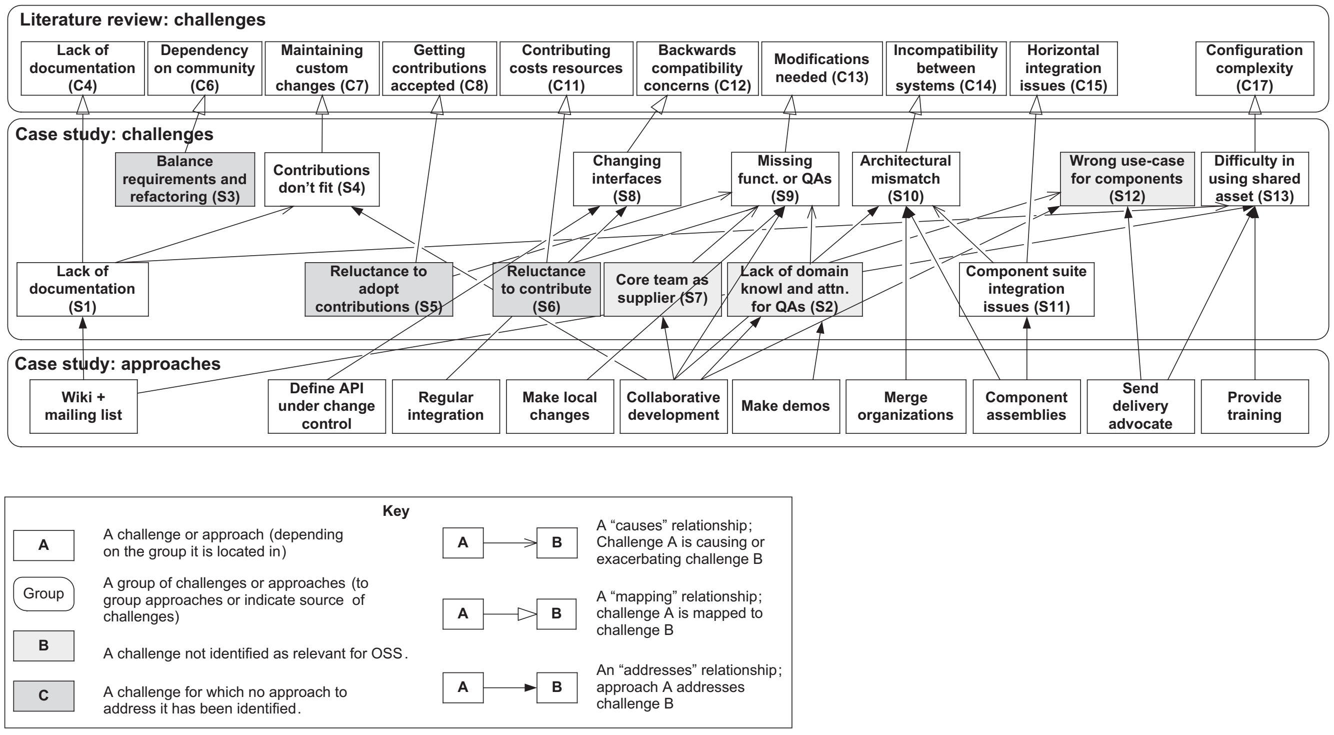
Literature review: challenges
Lack of documentation (C4)
Dependency on community (C6)
Maintaining custom changes (C7)
Getting contributions accepted (C8)
Contributing costs resources (C11)
Backwards compatibility concerns (C12)
Modifications needed (C13)
Incompatibility between systems (C14)
Horizontal integration issues (C15)
Configuration complexity (C17)
Case study: challenges
Balance requirements and refactoring (S3)
Contributions don't fit (S4)
Changing interfaces (S8)
Missing funct. or QAs (S9)
Architectural mismatch (S10)
Wrong use-case for components (S12)
Difficulty in using shared asset (S13)
Lack of documentation (S1)
Reluctance to adopt contributions (S5)
Reluctance to contribute (S6)
Core team as supplier (S7)
Lack of domain knowl and attn. for QAs (S2)
Component suite integration issues (S11)
Case study: approaches
Wiki + mailing list
Define API under change control
Regular integration
Make local changes
Collaborative development
Make demos
Merge organizations
Component assemblies
Send delivery advocate
Provide training
Key
A
A challenge or approach (depending on the group it is located in)
Group
A group of challenges or approaches (to group approaches or indicate source of challenges)
B
A challenge not identified as relevant for OSS.
C
A challenge for which no approach to address it has been identified.
A → B
A "causes" relationship; Challenge A is causing or exacerbating challenge B
A ▷ B
A "mapping" relationship; challenge A is mapped to challenge B
A ▶ B
An "addresses" relationship; approach A addresses challenge B

Fig. 5. Mapping of the challenges identified in the literature, the case study, and SoftCom's approaches to address them. Challenges coloured dark gray do not have associated approaches to address them. Challenges coloured light gray were identified in the case study but were not previously identified in the literature as a challenge with using OSS.

7.5.3. Maintaining custom changes (C7)

In both OSSD and ISSD, an integrator may choose to make custom changes. However, in both cases, it is preferred to give back those changes to the community/core team, in order to prevent additional efforts needed to maintain those custom changes. (Depending on whether an OSS integrator redistributes the changed software, it is in fact *required* to give back those changes, as is prescribed in OSS licenses.) In both cases, the integrator may take a different approach that is incompatible with the vision of the core team, resulting in contributions that do not fit.

7.5.4. Getting contributions accepted (C8)

OSS integrators have experienced issues in getting contributions to the OSS products that are integrated. This issue was also experienced in SoftCom. Various business divisions may want to make changes to the shared asset, but experience obstacles in getting their contributions accepted.

7.5.5. Contributing costs resources (C11)

An issue reported in relation to integrating OSS products is that contributing to an OSS project costs resources. In the case of OSS, organizations may not see the benefit, or just decide that it costs too many resources. In SoftCom, we also found that business divisions may be reluctant to contribute to the shared asset.

7.5.6. Backwards compatibility concerns (C12)

As OSS evolves, new versions are released. The speed with which an OSS product evolves depends on the OSS project's maturity and the liveliness of the community. Such changes could be product features, bug fixes and architectural changes. In SoftCom this challenge was manifested as changing interfaces of the shared asset. In development of products, business divisions need to choose for a certain version of the shared asset to use. Whenever there is a need to migrate to a newer version, interfaces may have changed, which results in additional cost to fix.

7.5.7. Modifications needed (C13)

When an organization integrates an OSS component into a product, it may have to make modifications by implementing missing functionality, or to make it fit within the architecture of the product [28]. A similar challenge was identified in the case study; a business division may find during integration that functionality is missing, or that quality requirements are not adequately addressed resulting in, for instance, poor performance.

7.5.8. System incompatibility (C14)

When integrating OSS products, each having been developed independently in a different context, incompatibility issues are bound to arise. The same challenge arises when software developed at other organizations (acquired through company acquisitions) is attempted to be integrated. Such software was developed in a certain business context, with certain requirements and restrictions. As a result, architectural mismatch [50] may occur, which is a common challenge for architects.

7.5.9. Horizontal integration issues (C15)

When integrating software components, different integration issues may arise. Horizontal integration, as opposed to vertical integration, refers to the integration of components on the same "level". (In vertical integration, software components would be built on top of each other, such as integrating an application on a particular runtime platform.) Horizontal integration issues could be as simple as syntactic mismatch (i.e., different implementation languages). In SoftCom, horizontal integration issues were manifested as integration issues of the component suite. Initially, the shared asset was a collection of components, and since there were no bounds in the ways these components could be integrated, integrators experienced significant integration issues.

7.5.10. Configuration complexity (C17)

The complexity of configuring OSS products can become a problem when using OSS products. We identified a similar challenge in SoftCom. Since the shared asset is to be used in different products, it needs to provide sufficiently abstract interfaces and use-case scenarios that fit different types of applications. This need for generic behavior causes that the shared asset becomes very difficult to use for the integrators.

7.6. Adopted approaches

SoftCom has adopted a number of approaches to address the identified challenges, as shown in the bottom layer in Fig. 5.

A number of these approaches are OSS practices that have been adopted, described in Section 6.1. Besides these practices, SoftCom has adopted a number of other approaches to address those challenges. We speculate that other organizations that have implemented (or plan to implement) an Inner Source model, as well as OSS projects, could benefit from such approaches. We address them next.

7.6.1. Wiki and mailing lists

In order to address a lack of documentation, SoftCom has set up an internal wiki and mailing list. The wiki facilitates knowledge sharing between the core team, which supplies the shared asset, and the business divisions that integrate the shared asset. A member of the core team described:

> "*What we've been working on for the last 2–3 years is setting up a wiki around everything we do and have in [our platform]. Our developers write articles on how to use the products, and that's really appreciated.*" —*P10, core team.*

While the wiki has proven to be an improvement in sharing knowledge, this is still a challenge. It is difficult to transfer knowledge, and there remains to be a "gap", according to one member of a business division:

> "*The core team has tried to solve that by setting up a wiki, and that does help us, but it's only one step. It was significant, because there was a huge gap, but there is still a gap.*" —*P11, business division.*

Besides a wiki, there are also mailing lists, which facilitate direct communication between customers (of the shared asset) and developers of the core team. In particular, developers can ask concrete questions regarding particular issues. A member of the core team described this as follows:

> "*[...] and we also set up mailing lists for concrete questions, like, 'hey, this is my problem'. We have set up a community to answer those types of questions, and our developers and architects follow these lists, and if they recognize a question they'll answer it, and that works quite well.*" —*P4, core team.*

7.6.2. Define API under change control

By more explicitly defining and managing a public API under change control, the interfaces of the shared asset will be more stable, which will reduce integration efforts when a business division migrates to a newly released version of the shared asset. One participant explained:

> "*In the past there was an extensive collection of private interfaces that you had to use to get things working, and the [business divisions] have been urging [the core team], like, guys, that drives us*

nuts, we need to define a public interface and put it under change control." —P3, business division.

7.6.3. Regular integration

The core team is also making demos with the shared asset, so that it is forced to play the role of integrator. As such, it is likely to encounter integration issues that can be subsequently addressed. This will help to overcome a lack of attention for both functional and non-functional requirements, since integration issues will be detected earlier and can be overcome in a more timely manner. One participant describes how his development team stays close to the latest version of the shared asset:

"*What [our development team] does, much more than other teams, is to integrate with work-in-progress versions of the platform. We are very close on the latest development, close to the [core] team, and collaborate well together. Every three weeks we take their build.*" —P3, business division.

7.6.4. Make local changes

The Inner Source model allows business divisions to make custom, "local" changes to their copy of the shared asset, if necessary. In certain cases, it may be desirable that a business division has this option, as one participant explained:

"*In the ideal case you're sure that it works, and if you're not, then you hope you'll know soon enough, so you can test it and show that it doesn't work what you had agreed. That way, we get the chance to solve and repair it. That's how you'd like to do it in the normal case. But, sometimes that doesn't work because a week before the release you find out that it might take two weeks to solve the problem. Well, then you don't have that time because the component happens to be developed in Bangalore [India], and this week turns out to be a celebration week. So, if you give the option to the business division to solve the problem themselves at such a late stage, in their own repository, then they'll need to have access to our source code, and then you'll have to go to an open source way of working.*" —P6, core team.

7.6.5. Collaborative development

Collaborative development is a project-based collaboration between the core team and a business division to develop new (or enhance existing) components. This is SoftCom's approach to "Open Source" style development. Rather than letting anybody contribute to the source code immediately, there must be a certain level of control. In collaborative development, the business division provides the expertise to make sure that the component implements the right functionality (and with sufficient attention for quality requirements), while the core team can ensure that the component adheres to the general architecture of the shared asset. One participant explained this interaction as follows:

"*What we do lately is, if we really need new functionality, we're doing some kind of collaborative development. We send someone to [the core team], who helps with the design and the development of the requirements that we set for the component. That person that is lend out to the [core team] really has knowledge of [our field], and by doing so we basically secure that the decisions that are made are the right ones.*" —P11, business division.

7.6.6. Make demos

The core team that develops the shared asset does not integrate its own product. This means that they have little experience with the ease of use of the shared asset's integration. Business divisions must find out how to best use the shared asset, and may encounter a variety of issues that were not anticipated. In order to address this, the core team now makes regular demo programs, which uses the shared asset. This way, the core team can experience first hand what potential issues may arise during integration. One participant described this as follows:

"*The core team often thinks they're done if the software just works. But precisely those extra things, such as the ease of use [of the shared asset] in a product development, receives limited attention. And what they introduced, a bit under our pressure [of the technology office], is that [the core team is] making regular demo versions to show what they've done.*" —P5, technology office.

7.6.7. Merge organizations

In order to speed up the architectural changes that are necessary to integrate newly acquired software into the shared asset, the new division was integrated with the core team on an organizational level. One participant who was closely involved in this described:

"*We acquired a company that delivers an information and communication system, which was sold on a pay-per-use contract. The software was specially developed to support that business model. That was a very different architecture than we were using. In order to speed up the adaptation of our architecture, we merged the two different organizations. We said, the way that information and communication system works is fundamentally so different, but it would be good to adopt that into our platform. And you can only do that by having the development done in one organization.*" — P6, core team.

7.6.8. Component assemblies

To address integration problems with the component-suite based model of the shared asset, the core team has started to offer the shared asset as sets of half-products, called 'assemblies'. These are pre-constructed sets of components that are already integrated and tested, thereby reducing integration and test efforts. A business division may deconstruct such an assembly and replace a component with a different one, if necessary. One participant described this as follows:

"*Take the LEGO® instruction booklet, and on page 8 there is half of the product that you need to build. That's what we deliver... we ruined the fun for you as a kid, we just pre-assembled it but we didn't glue it together, so you can still disassemble it if you want. And the advantage is that you don't have to assemble it anymore, which saves time, you don't have to test it anymore. That's more or less the parallel.*" —P6, core team.

7.6.9. Send delivery advocate

The core team can send a 'delivery advocate' to a business division. Delivery advocate is one of a few roles identified in [24] and is a member of the core team that assists a business division to integrate the shared asset. By being physically present as a local helpdesk, it becomes easier to help business divisions to integrate the shared asset. Naturally, this works better for business divisions that are located near the core team than for those located in different countries. One participant explained:

"*I think that certainly we, but also the business divisions, underestimated the importance of early feedback. And with the current projects that's going better. We send our people with the platform delivery, and they set up an integration desk at the customer, and if they start using the platform, then you'll know quickly whether it works or not. And then you're physically present to see whether* *it's really an issue, or whether the misconfigured the product."* —P6, core team.

7.6.10. Provide training

The core team can provide training. Architects from the core team explain the principles and rationale behind the architecture. Training is provided to business divisions in order to give them insights into the design of the shared asset, which will make it easier to understand how the shared asset can be used in a product. One participant of the core team explains:

"*We also give training in the area of, what is [the shared asset], and how its architecture was designed; how was it constructed, and why. Usually, our lead architect gives an introduction during the morning, followed by training focused on various subparts of the [shared asset].*" —P6, core team.

This is quite a traditional way of transferring knowledge, and is also provided by so-called commercial OSS providers, such as Red Hat, which sells support for Linux distributions and JBoss Enterprise Middleware software.

8. Discussion

In this section, we discuss the main findings from this research and compare some findings to related work (Section 8.1). Based on our findings, we draw the key implications of the reported work for the research and practice of developing software-intensive systems with OSS and ISS. This discussion is focused on the challenges and the approaches used, and which challenges have not been addressed.

8.1. Comparison of findings to related work

This paper reports on one organization ("SoftCom") that has adopted a project-based Inner Source initiative. One of the main objectives of this study was to identify the key challenges related to integration of a shared asset (the Inner Source Software). While our study is the first to focus on identifying such kinds of challenges, some other studies have also reported experiences. In this section, we compare our findings to these experience reports as far as they reflect on integration of the shared asset.

8.1.1. Comparison of challenges

In [5], Gurbani et al. report on their experiences with the Inner Source initiative at Alcatel-Lucent (which they refer to as "Corporate Open Source"), as well as lessons learned from these experiences. We observed a few commonalities and differences between their lessons and our results. We discuss these next.

8.1.1.1. Balancing refactoring and requirements (S3). We identified a challenge of keeping a balance between refactoring of the shared asset and fulfilling requirements (Section 7.2). Gurbani et al. reported on a similar lesson learned; they noted that: *"it is essential to recognize and accommodate the tension between cultivating a general, common resource on the one hand, and the pressure to get specific releases of specific products out on time."* [5]. The core team is responsible for delivering the shared asset (the common resource) and maintaining the conceptual and architectural integrity (which includes refactoring). In both cases, the core teams seem to be under pressure to deliver new features on the one hand, and maintain the shared asset's quality on the other hand.

8.1.1.2. Contributions don't fit (S4). SoftCom is a relatively large organization with many business divisions, all of which use the shared asset. A problem that was quite prevalent until a few years ago was that business divisions would develop contributions, which did not adhere to the architectural design principles of the shared asset, resulting in a misfit. The software may be quite useful for the business division involved, but would not be suitable for other customers as it was too specific. A similar problem also was reported by Gurbani et al. [5]: *"One of the most basic problems that many interviewees experienced was that developers were unaccustomed to thinking and designing solutions that were more general than their own product line."*

8.1.1.3. Reluctance to adopt contributions (S5). We found that the core team (at SoftCom) was somewhat reluctant to accept contributions. The core team is responsible for the design and maintenance of the shared asset; therefore, this reluctance may be fed by a "not-invented here" feeling as well as the obligation for further maintenance of code written by others. The interaction in the Inner Source project at Alcatel-Lucent seems to have been more open, and closer to the "Open Source" paradigm; Gurbani et al. [5] report that "*It is very unlikely for a developer in the [core team] to be cognizant of a feature being put into the code by another organization.*" This implies that it is easier for other developers (not members of the core team) to make changes to the shared asset directly. This is an essential difference with the situation at SoftCom, where it is explicitly not the case that non-core team members can make changes to the shared asset's code directly. Instead, contributions are much more controlled through the collaborative development mechanism.

There are a number of possible explanations for this difference. Firstly, the Inner Source project at Alcatel-Lucent was a new product and implementing a rapidly evolving technology, whereas the shared asset at SoftCom started as a well-established component suite with well-defined interfaces and functionality. Secondly, there is a significant difference in the size of the shared assets at Alcatel-Lucent and SoftCom. The former was reported to count approximately 48 thousand lines of code (in 2005 [5]), whereas the latter consisted of several millions lines of code, and is therefore much more complex and serving a larger variety of business needs. Therefore, contributing to such a large shared asset is naturally more complex.

8.2. Challenges lead to other challenges

We observe that certain challenges have led to other challenges; these are shown at the bottom in the middle layer (titled 'case study: challenges') in Fig. 5. Each of these "root" challenges cause or exacerbate other challenges (indicated by pointy arrows between challenges in the middle layer). We suggest that giving priority to these challenges while defining strategies to address them will have a positive, cascading effect. By addressing these root challenges, defined strategies may also indirectly address non-root challenges. We note that a challenge is not necessarily either a root challenge or "caused" (or exacerbated) by a root challenge; challenge S3, the tension of balancing implementing requirements and performing architectural refactoring, is neither caused by any other challenge, nor causing (or exacerbating) other challenges.

8.3. Unaddressed challenges

The three dark-gray colored challenges (S3, S5, S6) in Fig. 5 do not have associated approaches. We note that two of them are related to contributing (S5: core team's reluctance to adopt contributions, and S6: business divisions' reluctance to contribute). These challenges are two sides (sending and receiving) of the same medal, namely contribution, one of the core practices in OSSD. Both these challenges exacerbate the problem of missing functionality or insufficiently achieved quality requirements (S9). We assert that by improving the contribution mechanism, these two challenges can be addressed. As a result, this will also improve the level of mutual knowledge sharing within the organization, thereby addressing the lack of domain knowledge in the core team.

By improving the contribution mechanism in Inner Source, the pressure on the core team to fulfill all requested requirements may also be decreased (challenge S3: balance requirements and refactoring), which will allow them to allocate more time to perform maintenance and architectural refactoring.

8.4. Open research questions

Our research results have resulted in new insights but at the same time, it has identified a number of open research questions. We discuss them next.

8.4.1. Improving interaction and contributions

In Inner Source, business divisions have the right and means to make local changes to the software that is managed as an OSS asset, if so required. This partly addresses the problem that the core team, which manages the shared asset, lacks domain knowledge about certain requirements that a business division may have. This is especially useful if a business division is working towards a product release, and functionality turns out to be missing. Such changes should be given back to the core team, so as to take advantage of the Open Source paradigm. However, our findings suggest that this rarely happens. The core team heavily guards the shared asset's architecture, and is reluctant to accept the maintenance responsibility of code that was 'not developed here'. One common concern is that contributions may not respect the architectural principles of the shared asset. This challenge may also arise in OSS development: a case study reported in [51] showed that after two years of development, the actual architecture differed significantly from the conceptual (designed) architecture. This phenomenon is also referred to as 'architectural drift'. Linus Torvalds, creator and chief architect of the Linux kernel project, recently expressed that he was not pleased with the current state of the implementation [52].

One approach SoftCom is taking to address this challenge is to do collaborative development. However, our results suggest that the organization would benefit if the contribution mechanism would be improved and better exploited. It would be valuable to improve our understanding of how business divisions can make higher quality contributions that can be more easily accepted by the core team.

8.4.2. Requirements versus refactoring

Since various business divisions use the shared asset, the number of requests to the core team for functionality and improvements can be quite high. Furthermore, besides the need to prioritize the business divisions' requests, the core team also needs to maintain the soundness of the shared asset's architecture. Therefore, there is a continuous need to balance the tension between fulfilling business divisions' requirements on the one hand, and performing architectural refactoring on the other. We assert that it would be very valuable to gain a deeper insight into what lessons can be learned here from the OSS paradigm.

8.4.3. Improving knowledge sharing

One of the most recurring challenges we have identified is that business divisions found it difficult to use the shared asset. That is, business divisions have great difficulty in building an application based on the shared asset. Developers and architects have a strong need for 'how-to' knowledge, how to use the component. Though this challenge has been partly addressed by the set up of a wiki and a mailing list through which knowledge can be shared that has increased the liveliness of the community. These measures may not to be sufficient. Hence, transferring knowledge effectively among different development teams remains to be a challenge.

8.5. Limitations of this study

We are aware of a few limitations of this study that we discuss in this section. They are classified in limitations regarding construct validity, external validity and reliability. Since this case study has an exploratory nature, internal validity is not a concern, as there are no claims about causal relationships [15]. (We note that the "causal" relationships between the "root" challenges (see Sections 7.5 and 8.1) that cause or exacerbate other challenges are not a matter of *internal* validity, but rather of *reliability*; the relationships were part of the findings, rather than being tested in this research. Threats to reliability are discussed below.)

8.5.1. Construct validity

Construct validity is concerned with the question whether the researcher measures what he or she intends to measure. This study is limited since we have gathered data from only one source (interviews). The number of interviews is limited to 11. However, we found that all participants informed us with more or less the same description of their experiences, which hinted at data saturation [53]. We interviewed people from different divisions, each expressing their experiences and views, thereby providing us with a rounded view of the topic at hand. We found that results of all interviews were consistent, which increases our confidence in the trustworthiness of the data. The consistency of our data gives us confidence that we have identified real challenges that practitioners experience. All challenges can be traced back to at least one participant, and 10 out of 13 challenges were mentioned by two or more participants.

8.5.2. External validity

A commonly expressed concern of case study methodology is that no statistical generalization can be achieved [53]. However, the goal of case study research is not to achieve statistical generalization, but rather an analytical generalization. This is of particular importance for studying a phenomenon such as Inner Source, since each case of it is tailored to the organization in which it is implemented. Another organization that has adopted different OSS practices is likely to encounter different challenges. This study is a first attempt to bring clarity about a relatively unexplored area. In Section 6 we presented a high-level overview of the Inner Source implementation in SoftCom, which provide context to interpret our findings presented in Section 7. Also, our case study was performed at an organization that has adopted a project-based Inner Source initiative. Different challenges may occur in an infrastructure-based Inner Source initiative.

8.5.3. Reliability

Reliability is the level to which the operational aspects of the study, such as data collection and analysis procedures, are repeatable with the same results. Given that the first author conducted the primary initial interpretation of the data, the issue of interpretive validity and trustworthiness of the analysis bears consideration. This issue was addressed by following three common procedures to establish validity in qualitative projects [54], which have been briefly discussed in Section 5.3. We describe these in more detail below.

8.5.3.1. Triangulation. The first procedure is triangulation, which is validity procedure to search for convergence among multiple and different sources of information [54]. There are four types of triangulation: across data sources (such as participants), theories, methods (such as data collection methods), and among different investigators [54]. In this research, we have applied triangulation across data sources, as we interviewed 11 participants and analyzed and cross-compared their "story lines". Through this procedure, we found that challenges and approaches were consistently described. A second form of triangulation that we performed is among different investigators. In several face-to-face meetings, two researchers have extensively discussed the study context, findings and conclusions.

8.5.3.2. Audit trail. Secondly, we provide a traceable, documented justification of the process by which research conclusions were reached, thus providing an *audit trail* of the process, as recommended by Guba [55]. All interviews were recorded and transcribed verbatim in order to make sure that no data reduction occurred prematurely. The transcription of the interviews was done by a single researcher, and was not crosschecked by others. This could potentially have resulted in information loss. However, we believe this risk was minimized, as all transcripts were compared once more to the audio recordings, in order to make sure that nothing was lost during transcription. A sample of the memoing and coding process is provided in Section 5.3 above.

8.5.3.3. Member checking. The third validity procedure is *member checking*, whereby data and interpretations are taken back to the participants to allow them to confirm the credibility [54]. We used a form of member checking whereby interviewees were subsequently provided with an initial draft of this paper. Furthermore, after analyzing our data, we became aware of a number of deliverables of a research project that SoftCom had been involved in, that we had access to. Some of the participants of our study had been involved in authoring these deliverables, which is why this is a form of member checking. These deliverables contained descriptions of a few of the challenges and approaches that we have identified and presented in Section 7 and therefore further confirmed our findings.

9. Conclusion and future work

A number of large organizations have recognized the successful mode of software development that occurs within Open Source Software development, and have adopted OSSD practices within their organization's boundaries. While different terms have been used to describe this phenomenon, in this paper we use the term "Inner Source", and have termed the produced software in such an environment "Inner Source Software" (ISS).

There have been various experience reports of organizations that have adopted an Inner Source approach. These studies typically report on experiences of adopting OSSD practices as well as encountered challenges and lessons learned, and as such, present rather general accounts of adopting OSSD practices (i.e., Inner Source).

In this paper, we have focused on challenges in Inner Source from a software integrator's perspective. We report on an in-depth exploratory case study at a large organization that has adopted a number of OSSD practices for its in-house software development. In this paper we have explicitly linked development of products with OSS (which has been studied extensively) and development of products with ISS (which is a relatively new field of research). One significant difference is that, in the one case (OSS), the software was developed by an external, unknown workforce, whereas in the other (ISS), the software was developed by an internal, known workforce. This means that in the one case, development of components is out of the organization's control, whereas in ISS, the "OSS community" is grown within the organization. Despite this difference we have observed that there are many common challenges.

The findings of this study provide valuable insights to organizations that may wish to adopt Inner Source by informing them about challenges that have been experienced by a large organization. This paper has also identified some approaches that an organization can use to address the challenges identified by this case study. Other organizations that are experiencing similar challenges may learn from these approaches. Furthermore, this paper has also identified a number of open research questions that can help researchers to form a research agenda.

Organizations can benefit greatly from adopting OSS development practices, however, more research is needed to fully understand how to address the challenges involved in OSSD. We believe that Inner Source can provide opportunities for an organization to improve its software development processes. In particular, it would be quite valuable to increase our understanding of how Inner Source can facilitate a higher degree of interaction among business divisions within an organization, in terms of contributions both to the shared asset and architectural and sharing knowledge.

This paper contributes to the literature by documenting the challenges involved in developing and using ISS and approaches to address those challenges. This can be of interest to organizations that wish to adopt OSSD practices. These new insights can be combined with the insights reported so far in other studies (e.g., [4–6]). In particular, we are planning to continue our research efforts to provide practical, empirical-based guidelines that will give organizations insight into what practices are appropriate, and how particular challenges can be addressed by an Inner Source approach.

Acknowledgements

This work is partially funded by IRCSET under grant no. RS/2008/134 and by Science Foundation Ireland grant 03/CE2/I303_1 to Lero (www.lero.ie). We are grateful to the participants of our study for their time and enthusiasm. We thank the anonymous reviewers for their useful comments and suggestions to improve this paper, particularly Sections 5–8. The first author is grateful to Sarah Beecham for her valuable feedback on earlier drafts of Fig. 2.

References

[1] C. Melian, Progressive Open Source: The construction of a development project at Hewlett-Packard, PhD dissertation, Stockholm School of Economics, 2007.
[2] J. Robbins, Adopting Open Source Software Engineering (OSSE) practices by adopting OSSE tools, in: J. Feller, B. Fitzgerald, S.A. Hissam, K.R. Lakhani (Eds.), Perspectives on Free and Open Source Software, MIT Press, 2005, pp. 245–264.
[3] E.S. Raymond, The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary, O'Reilly Media, 2001.
[4] J. Wesselius, The bazaar inside the cathedral: business models for internal markets, IEEE Software 25 (2008) 60–66, doi:10.1109/MS.2008.79.
[5] V.K. Gurbani, G. Anita, J.D. Herbsleb, A case study of a corporate open source development model, in: Proceedings of the 28th International Conference on Software Engineering, Shanghai, China, 2006, pp. 472–481, 10.1145/1134285.1134352.
[6] J. Dinkelacker, P.K. Garg, R. Miller, D. Nelson, Progressive open source, in: Proceedings of the 24th International Conference on Software Engineering, Orlando, FL, USA, 2002, pp. 177–184, 10.1145/581339.581363.
[7] T. O'Reilly, Lessons from open source software development, Communications of the ACM 42 (1999) 33–37, doi:10.1145/299157.299164.
[8] A. Mockus, J.D. Herbsleb, Why not improve coordination in distributed software development by stealing good ideas from open source?, in: Proceedings of the 2nd Workshop on Open Source Software Engineering, Orlando, FL, USA, 2002, pp. 35–37.
[9] J. Erenkrantz, R.N. Taylor, Supporting distributed and decentralized projects: drawing lessons from the open source community, in: Proceedings of the 1st Workshop on Open Source in an Industrial Context, Anaheim, California, 2003.

[10] J. Asundi, Software engineering lessons from open source projects, in: J. Feller, B. Fitzgerald, A. van der Hoek (Eds.), Proceedings of the 1st Workshop on Open Source Software Engineering, Toronto, ON, Canada, 2001.
[11] J. Lindman, M. Rossi, P. Marttiin, Applying open source development practices inside a company, in: B. Russo, E. Damiani, S. Hissam, B. Lundell, G. Succi (Eds.), Open Source Development, Communities and Quality, Springer, 2008, pp. 381–387, doi:10.1007/978-0-387-09684-1_36.
[12] D. Riehle, J. Ellenberger, T. Menahem, B. Mikhailovski, Y. Natchetoi, B. Naveh, T. Odenwald, Open collaboration within corporations using software forges, IEEE Software 26 (2009) 52–58, doi:10.1109/MS.2009.44.
[13] K. Martin, B. Hoffman, An open source approach to developing software in a small organization, IEEE Software 24 (2007) 46–53, doi:10.1109/MS.2007.5.
[14] K. Stol, M. Ali Babar, Challenges in using open source software in product development: a review of the literature, in: Proceedings of the 3rd Workshop on Emerging Trends in FLOSS Research and Development, ACM, Cape Town, South Africa, 2010, pp. 17–22, 10.1145/1833272.1833276.
[15] R.K. Yin, Case Study Research: Design and Methods, 3rd ed., Sage Publications, Thousand Oaks, CA, 2003.
[16] K. Petersen, C. Wohlin, A comparison of issues and advantages in agile and incremental development between state of the art and an industrial case, Journal of Systems and Software 82 (2009) 1479–1490, doi:10.1016/j.jss.2009.03.036.
[17] G. Gaughan, B. Fitzgerald, M. Shaikh, 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 Patras, Greece, 2009, pp. 20–27, 10.1109/SEAA.2009.86.
[18] F. Van der Linden, Applying open source software principles in product lines, UPGRADE 10 (2009) 32–41.
[19] Ø. Hauge, C. Ayala, R. Conradi, Adoption of open source software in software-intensive organizations – a systematic literature review, Information and Software Technology 52 (2010) 1133–1154, doi:10.1016/j.infsof.2010.05.008.
[20] T. Østerlie, L. Jaccheri, A critical review of software engineering research on open source software development, in: Proceedings of the 2nd European Symposium on Systems Analysis and Design, Gdansk, Poland, 2007, pp. 12–20.
[21] J. Feller, B. Fitzgerald, Understanding Open Source Software Development, Pearson Education Ltd., 2002.
[22] K. Stol, M. Ali Babar, B. Russo, B. Fitzgerald, the use of empirical methods in open source software research: facts, trends and future directions in: Proceedings of the 2nd Workshop on Emerging Trends in FLOSS Research and Development, IEEE, Vancouver, Canada, 2009, pp. 19–24, 10.1109/FLOSS.2009.5071355.
[23] M. Theunissen, D. Kourie, A. Boake, Corporate-, agile- and open source software development: a witch's brew or an elixir of life?, in: B. Meyer, J.R. Nawrocki, B. Walter (Eds.), Balancing Agility and Formalism in Software Engineering, Springer-Verlag, 2008, pp. 84–95, doi:10.1007/978-3-540-85279-7_7.
[24] V.K. Gurbani, A. Garvert, J.D. Herbsleb, Managing a corporate open source software asset, Communications of the ACM 53 (2010) 155–159, doi:10.1145/1646353.1646392.
[25] C. Melian, M. Mähring, Lost and gained in translation: adoption of open source software development at Hewlett–Packard, in: B. Russo, E. Damiani, S. Hissam, B. Lundell, G. Succi (Eds.), Open Source Development, Communities and Quality, Springer, 2008, pp. 93–104, doi:10.1007/978-0-387-09684-1_8.
[26] P. Clements, L. Northrop, Software Product Lines: Practices and Patterns, Addison-Wesley, 2002.
[27] K.C. Wallnau, S.A. Hissam, R.C. Seacord, Building Systems from Commercial Components, Addison-Wesley, 2002.
[28] K. Ven, H. Mannaert, Challenges and strategies in the use of open source software by independent software vendors, Information and Software Technology 50 (2008) 991–1002, doi:10.1016/j.infsof.2007.09.001.
[29] B. Kitchenham, S. Charters, Guidelines for performing systematic literature reviews in software engineering, Technical report No. EBSE 2007-001, 2007.
[30] Ø. Hauge, C.-F. Sørensen, A. Røsdal, Surveying industrial roles in open source software development, in: J. Feller, B. Fitzgerald, W. Scacchi, A. Sillitti (Eds.), Open Source Development, Adoption and Innovation, Springer, 2007, pp. 259–264, doi:10.1007/978-0-387-72486-7_25.
[31] C. Ayala, Ø. Hauge, R. Conradi, X. Franch, J. Li, K.S. Velle, Challenges of the open source component marketplace in the industry, in: C. Boldyreff, K. Crowston, B. Lundell, A.I. Wasserman (Eds.), Open Source Ecosystems: Diverse Communities Interacting, Springer, 2009, pp. 265–271, doi:10.1007/978-3-642-02032-2_19.
[32] J. Merilinna, M. Matinlassi, State of the art and practice of open source component integration, in: Proceedings of the 32nd Euromicro Conference on Software Engineering and Advanced Applications, IEEE Computer Society, 2006, pp. 170–177, 10.1109/EUROMICRO.2006.61.
[33] W. Chen, J. Li, J. Ma, R. Conradi, J. Ji, C. Liu, An empirical study on software development with open source components in the chinese software industry, Software Process: Improvement and Practice 13 (2008) 98–100, doi:10.1002/spip.361.
[34] P. Conlon, P. Carew, A risk driven framework for open source information systems development, in: M. Scotto, G. Succi (Eds.) Proceedings of the 1st International Conference on Open Source Systems, Genova, Italy, 2005, pp. 200–203.
[35] J. Krivoruchko, The use of open source software in enterprise distributed computing environments, in: J. Feller, B. Fitzgerald, W. Scacchi, A. Sillitti (Eds.), Open Source Development, Adoption and Innovation, Springer, 2007, pp. 277–282, doi:10.1007/978-0-387-72486-7_28.
[36] F. Tiangco, A. Stockwell, J. Sapsford, A. Rainer, E. Swanton, Open-source software in an occupational health application: the case of Heales Medical Ltd., in: Proceedings of the 1st International Conference on Open Source Systems, 2005, pp. 130–134.
[37] C. Bac, O. Berger, V. Deborde, B. Hamet, Why and how to contribute to libre software when you integrate them into an in-house application? in: M. Scotto, G. Succi (Eds.) Proceedings of the 1st International Conference on Open Source Systems, Genova, Italy, 2005, pp. 113–118.
[38] J. Akkanen, H. Demeter, T. Eppel, Z. Ivánfi, J. Nurminen, P. Stenman, Reusing an open source application — practical experiences with a mobile CRM pilot, in: J. Feller, B. Fitzgerald, W. Scacchi, A. Sillitti (Eds.), Open Source Development, Adoption and Innovation, Springer, 2007, pp. 217–222, doi:10.1007/978-0-387-72486-7_18.
[39] T.R. Madanmohan, R. De', Open source reuse in commercial firms, IEEE Software 21 (2004) 62–69, doi:10.1109/MS.2004.45.
[40] C. Ayala, C. Sørensen, R. Conradi, X. Franch, J. Li, Open source collaboration for fostering off-the-shelf components selection, in: J. Feller, B. Fitzgerald, W. Scacchi, A. Sillitti (Eds.), Open Source Development, Adoption, Innovation, Springer, 2007, pp. 17–30, doi:10.1007/978-0-387-72486-7_2.
[41] A. Jaaksi, Experiences on Product Development with Open Source Software, in: Open Source Development, Adoption and Innovation, Springer, 2007, pp. 85–96, doi:10.1007/978-0-387-72486-7_7.
[42] K. Ven, J. Verelst, The importance of external support in the adoption of open source server software, in: C. Boldyreff, K. Crowston, B. Lundell, A.I. Wasserman (Eds.), Open Source Ecosystems: Diverse Communities Interacting, Springer, 2009, pp. 116–128, doi:10.1007/978-3-642-02032-2_12.
[43] H. Mannaert, K. Ven, The use of open source software platforms by independent software vendors: issues and opportunities, in: Proceedings of the 5th Workshop on Open Source Software Engineering, ACM, 2005, doi:10.1145/1082983.1083266.
[44] C. Ruffin, C. Ebert, Using open source software in product development: a primer, IEEE Software 21 (2004) 82–86, doi:10.1109/MS.2004.1259227.
[45] P.J. Ågerfalk, A. Deverell, B. Fitzgerald, L. Morgan, Assessing the role of open source software in the European secondary software sector: a voice from industry, in: M. Scotto, G. Succi (Eds.) Proceedings of the 1st International Conference on Open Source Systems, Genova, Italy, 2005, pp. 82–87.
[46] A.C. Edmondson, S.E. McManus, Methodological fit in management field research, Academy of Management Review 32 (2007) 1155–1179.
[47] J.M. Verner, J. Sampson, V. Tosic, N.A. Abu Bakar, B.A. Kitchenham, Guidelines for industrially-based multiple case studies in software engineering, in: Proceedings of the Third International Conference on Research Challenges in Information Science, 2009, pp. 313–324, 10.1109/RCIS.2009.5089295.
[48] S.J. Taylor, R. Bogdan, Introduction to Qualitative Research, John Wiley & Sons, New York, 1984.
[49] C.B. Seaman, Qualitative methods in empirical studies of software engineering, IEEE Transactions on Software Engineering 25 (1999) 557–572, doi:10.1109/32.799955.
[50] D. Garlan, R. Allen, J. Ockerbloom, Architectural mismatch: why reuse is so hard, IEEE Software 12 (1995) 17–26, doi:10.1109/52.469757.
[51] E. Nakagawa, E. de Sousa, K. de Brito Murata, G. de Faria Andery, L. Morelli, J. Maldonado, Software architecture relevance in open source software evolution: a case study, in: Proceedings of the 32nd International Computer Software and Applications Conference, IEEE Computer Society Washington, DC, USA, 2008, pp. 1234–1239, 10.1109/COMPSAC.2008.171.
[52] A. Modine, Linus calls Linux 'bloated and huge', in: The Register, 2009.
[53] P. Runeson, M. Höst, Guidelines for conducting and reporting case study research in software engineering, Empirical Software Engineering 14 (2009) 131–164, doi:10.1007/s10664-008-9102-8.
[54] J.W. Creswell, D.L. Miller, Determining validity in qualitative inquiry, Theory into Practice 39 (2000) 124–130.
[55] E. Guba, Criteria for assessing the trustworthiness of naturalistic inquiries, Educational Communication and Technology 29 (1981) 75–92.