Inner Source Project Management

Martin Höst, Klaas-Jan Stol, and Alma Oručević-Alagić

Abstract. Software development organizations are continuously looking for better ways to manage their projects. An emerging approach to achieve this is Inner Source, which refers to the adoption of Open Source development practices within the confines of an organization. With an Inner Source approach, individuals, teams, and departments within an organization can start software projects, very similar to the Open Source model. This affects the way projects are managed in a variety of ways. Firstly, it will affect strategic aspects such as a software sourcing strategy that includes decisions on which software can be "Inner-Sourced." Secondly, at the tactical level, organizations should choose an appropriate Inner Source adoption model that suits the goals and scope of the organization. Finally, it will affect the operational aspects of a project, for example, in the way different people across a whole organization can access the source code and make improvements. Furthermore, Inner Source makes communication much more transparent. While Inner Source offers a variety of potential benefits to an organization, there are also a number of challenges to address. This chapter discusses how the introduction of Inner Source may affect conventional software developing environments and especially how it affects software project management aspects. Based on our studies and those presented in the literature, it outlines a number of benefits of Inner Source as well as a number of challenges and some suggestions as to how they can be addressed.

Please cite as: M. Höst, K.J. Stol, and A. Orucevic-Alagic. Inner Source Project Management. In: G. Ruhe and C. Wohlin (eds.), *Software Project Management in a Changing World*, Springer-Verlag Berlin Heidelberg 2014, pp. 343-369

14.1 Introduction

The use of Open Source Software (OSS) in industry has seen an unprecedented growth during the last decade. Both the use of OSS development tools, such as Eclipse, and the inclusion of OSS components (e.g., the Apache web server[1]) and software frameworks (e.g., the Struts framework[2]) has increased. OSS refers to software that is distributed under an Open Source license,[3] which permits that the software's source code is freely available to anyone to change to his or her needs (while respecting the conditions of the license). Successful OSS projects are often developed by a large number of disparate, geographically spread, developers around the world, as outlined in Chap. 10.

The perceived success of OSS projects, and the ability to manage distributed development, has resulted in efforts to introduce Open Source development principles inside companies; this phenomenon is called "Inner Source" (Stol et al. 2014) though other terms have been used, such as "Corporate Open Source" (Gurbani et al. 2006) and "Progressive Open Source" (Dinkelacker et al. 2002). Stol et al. (2011) defined Inner Source as follows:

> Definition: 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.

Motivations for adopting this way of working include the inherent support for distributed development and the potential to increase software reuse and quality due to an increased availability, openness and transparency of the software, and an implied invitation to anyone in an organization to join development or contribute otherwise (Gaughan et al. 2009).

There are, however, a number of key differences between conventional in-house development and Open Source development. For example, Open Source projects typically do not have the same budget constraints and lead-time limitations that are typical issues for project management in traditional projects. Therefore, Inner Source initiatives always lead to a tailoring of Open Source development practices to fit a corporate context. The objective of this chapter is to present an overview of Inner Source development, and identify implications for typical project management issues.

The remainder of this chapter proceeds as follows:

- Section 14.2 outlines what Inner Source is by positioning it in the software development landscape with respect to other strategies. This section also presents a number of motivations, outlining why organizations adopt Inner Source. Furthermore, this section presents different adoption models as well as a number of new management roles that may emerge as a result of adopting Inner Source.

[1] http://httpd.apache.org/.
[2] http://struts.apache.org/.
[3] http://opensource.org/licenses/index.html.

- Section 14.3 presents a framework that identifies a number of key themes related to project management in which Inner Source and conventional project management approaches can differ. This framework serves as a "lens" for the remainder of the chapter, which presents two case studies.
- Section 14.4 presents the case studies that we conducted at two organizations. The first case study presents an analysis of how well an organization's current project management approach aligns with an Open Source development approach. This case study provides insights into some of the key changes that an organization may need to make in terms of project management issues, and as such represents a scenario prior to adopting Inner Source. The second case study presents an analysis of an organization that has adopted Inner Source for several years and discusses a number of project management challenges as well as approaches to how those challenges were addressed. As such, this case presents a scenario after adopting Inner Source.
- Section 14.5 discusses the findings of the two case studies presented in Sect. 14.4, as well as a number of implications for practice. Based on this, we present a number of recommendations for Inner Source project management. In this section, we also propose a number of new directions for future research.

14.2 Inner Source

This section presents an overview of Inner Source. We outline what Inner Source is by positioning it in terms of openness of the software product and software process (Sect. 14.2.1). This is followed by a discussion of why organizations would want to adopt Inner Source (Sect. 14.2.2). Section 14.2.3 discusses Inner Source adoption models, and Sect. 14.2.4 discusses a number of Inner Source project management roles.

14.2.1 Positioning Inner Source as a Strategy

As outlined above, the term Inner Source refers to the adoption of OSS development practices within an organization's boundaries. It is informative to discuss the implications of this in relation to other trends in the software landscape, such as open-sourcing (Ågerfalk and Fitzgerald 2008; Oručević-Alagić and Höst 2010), so as to clearly define the scope of this chapter. Inner Source can be characterized by two features: (1) the development process is opened up to the whole developer community within an organization; and (2) the resulting product is proprietary, and access to its source code is limited by an organization's boundaries. Figure 14.1 presents a typology using these two characteristics as dimensions.

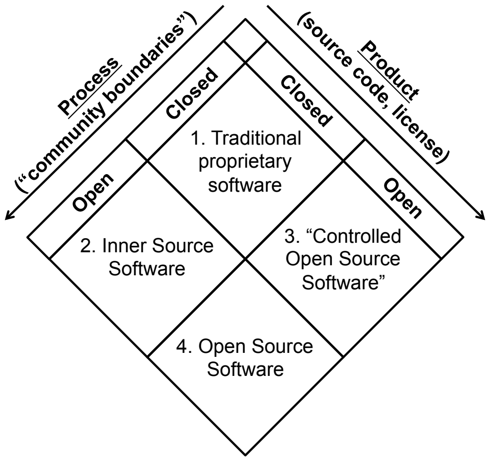
Process ("community boundaries")
Product (source code, license)
Closed
Open
Closed
Open
1. Traditional proprietary software
2. Inner Source Software
3. "Controlled Open Source Software"
4. Open Source Software

Fig. 14.1 Typology of open and closed software based on process and product dimensions

We refer to "conventional" development as proprietary software development, using a closed process, and a resulting closed product. A closed process in this context refers to the fact that there is no "open" development community in which developers can join as new members and start contributing to the development process as they see fit. Instead, in conventional software development, teams are predefined and often organized in a hierarchical fashion. The product is also closed, in that the source code is only accessible by the project team and not publicly available.

Open Source Software is at the other end in both dimensions, with an open process and an open product. A typical OSS project has an open process that welcomes new contributors (see also Chap. 13). In fact, an Open Source project's very success depends on how many developers it can attract. The product is open as well as the source code is necessarily available so as to comply with the requirements of an OSS license.

Products that are open but with a closed process is what we label "controlled Open Source Software." Such OSS projects comply formally with an OSI-approved Open Source license,[4] and as such are formally "Open Source." However, in practice, the development process is not open and the product's development is tightly controlled by one organization (or possibly a consortium), or a limited number of individuals. One such example is the Lua programming language,[5] which is designed, implemented, and maintained by a team of three researchers. Lua's source code is freely available under the MIT license. Ideas and suggestions are welcome in the project (and some of the project's key features originated from

[4] As of August 2013, there are 70 OSI-approved licenses.
[5] http://www.lua.org.

such suggestions), but write-access (or "commit-access") to the source code repository cannot be "earned" by others and is limited to the three original implementers (Ierusalimschy 2008). We note that controlled OSS is different from sponsored OSS, in which organizations contribute ("sponsor") code or other contributions or resources (Capiluppi et al. 2012). Sponsored OSS projects have an open process, where new developers are welcome to contribute, which differentiates it from "controlled" OSS.

Inner Source, then, is characterized by an open process, and a closed product. In this context, the process is only "open" to one organization (or a consortium of partners or subcontractors), and not to anyone outside the organization (or consortium). In principle, anyone in the organization is free to submit contributions. The resulting product, however, is still closed, in that it is proprietary and has no Open Source license. Licensing, while a common concern for organizations that wish to adopt OSS (Stol and Babar 2010), is therefore not a concern in Inner Source.

The typology in Fig. 14.1 is, of course, a simplification, a snapshot of reality at any one time and organizations may engage in more than one of the four scenarios. For example, there is an increasing involvement of organizations in sponsored Open Source (mentioned above), and products and services are increasingly based on such projects.

Over time, an organization may change its software sourcing strategy and move to another scenario within Fig. 14.1. For example, organizations may open source their product (Oručević-Alagić and Höst 2010), opening both the process and the product. This is, for example, what Netscape Corp. did in the late 1990s with their Netscape Navigator, from which the Mozilla web browser project emerged. Involvement of organizations may vary greatly, from being an "industry-led" project (Capiluppi et al. 2012) to total withdrawal of their involvement in the project. In a scenario where an organization is opening up its product from an Inner Source setting, we no longer speak of Inner Source, but of open-sourcing, and the resulting product becomes Open Source (Ågerfalk and Fitzgerald 2008; Oručević-Alagić and Höst 2010).

14.2.2 Motivations and Benefits of Inner Source

Organizations may adopt Inner Source for a variety of reasons and offers a number of benefits to organizations (Gaughan et al. 2009). We discuss some of these below:

- **Improved reuse.** An internal repository or "forge" that hosts Inner Source projects can provide a good starting point for projects, and as such increase reuse within the organization (Dinkelacker et al. 2002; Lindman et al. 2008; Vitharana et al. 2010).
- **Improved quality.** Inner Source projects can benefit from "Linus's Law," whereby a large community of developers peer review contributions. Since

contributions are under large-scale scrutiny, contributors may be aware of their reputation and be motivated to write “good” code (Dinkelacker et al. 2002; Riehle et al. 2009).

- **Rapid developer redeployment.** Since developers are familiar with a standard set of common tools and infrastructure used within an Inner Source setting, as well as with the available software on the internal forge, developers can be more easily transferred to other projects or products (Dinkelacker et al. 2002). This in turn will also reduce time-to-market, as project start-up time can be reduced.
- **Increased awareness.** An open environment facilitates an increased awareness of the software that is developed within an organization (Lindman et al. 2008, 2013).
- **Open innovation.** Besides an increased awareness among developers, a more open development environment may also support the concept of Open Innovation (Morgan et al. 2011; Lindman et al. 2013).
- **Large developers pool.** Open collaboration on software projects facilitates a large developer pool, thereby broadening the expertise of the developer community (Wesselius 2008; Riehle et al. 2009).
- **Increased development speed.** Given a larger development community, development of Inner Source projects may benefit from an increased development speed (Dinkelacker et al. 2002). A single team may not have enough capacity to implement all required functionality (Wesselius 2008). This in turn also supports a faster time-to-market.

The degree to which these benefits can be achieved will depend on how successful an Inner Source initiative is. While the benefits listed here have been reported in the extant literature, Inner Source as a field of research is still in its nascent phase, and precise predictions as to the extent to which benefits can be achieved cannot be given.

An important consideration for any organization with software as a product (or part thereof) is the strategic “make-or-buy” decision, or the sourcing strategy. Van der Linden et al. (2009) presented a “decision map,” which distinguishes differentiating software (software that provides unique business value) and commodity software, such as operating systems, database systems and compilers. According to this decision map, outsourcing “differentiating” software is considered unwise, since this is the (usually relatively small) part of a product that offers a competitive advantage. Commodity software, on the other hand, should typically not be built in- house, as this would waste costly resources. No organization should develop its own database system or operating system; these are commodities and should be acquired elsewhere. Inner Source, then, could be one strategy toward commodification. It is a suitable approach to develop software that is considered sufficiently valuable to develop in-house, while delivering functionality needed by different stakeholders.

One important software architectural strategy within which commodification of software is common is a software product line (Van der Linden 2009). Software product lines typically consist of a common platform on the one hand and a number of derived applications that are based on that platform. Van der Linden (2009) discussed how Inner Source can help to overcome a number of bottlenecks that are common in product lines, such as the dependencies of an application on the platform. Development of the platform may lag behind development of a platform-based application, possibly delaying its delivery to the market. Van der Linden (2009) reported that Inner Source adoption at Philips Healthcare has led to a reduction of the time-to-market of at least 3 months.

14.2.3 Inner Source Adoption Models

Organizations can adopt Inner Source in different ways. In fact, each Inner Source implementation must be tailored to the specific context of an organization (Gaughan et al. 2009). Various factors may influence how an organization implements its Inner Source initiative, such as its product domain, whether or not it is subject to any legal regulations, and an organization's size. However, despite this variety, Gurbani et al. (2010) identified two main models of Inner Source adoption: the infrastructure-based model and the project-specific model. These are briefly discussed below.

14.2.3.1 Infrastructure-Based Model

The infrastructure-based model is the simplest model. In this setting, an organization facilitates Inner Source projects by providing the necessary infrastructure such as code repositories, bulletin board software, and mailing list servers. These tools enable project teams and individual developers to host an Inner Source project, very similar to how individuals can publish their software on one of the public code repositories, such as SourceForge.net. SAP (Riehle et al. 2009), Hewlett-Packard (Dinkelacker et al. 2002), and Nokia (Lindman et al. 2008) are organizations that have adopted this model.

14.2.3.2 Project-Specific Model

The project-specific Inner Source model is a more strategic approach and builds on the infrastructure-based model. In this setup, there is one dedicated division (project team or product group) that takes responsibility for development, maintenance, and support of an Inner Source project, referred to as a shared asset. This division is sometimes referred to as the "core team," similar to a core team in an OSS project. The project-specific model focuses on strategic reuse of the shared asset. A key responsibility of the core team is to provide ongoing support to customers of the Inner Source project. Since this requires dedicated resources to sustain the Inner Source initiative, an organization may have to choose carefully which projects

Table 14.1 Key differences between infrastructure-based and project-specific inner Source models (adapted from Stol et al. 2011)

Characteristic	Infrastructure-based	Project-specific
Software reuse	Opportunistic and ad hoc. Optimize for sharing maxi- mum number of projects	Strategically planned. Optimize for sharing critical assets
Support	Optional, depends on success of project	Essential for project success. Needs organizational support and funding
Owner/ maintainer	Individual or team who created the project	Central "core team." Needs orga- nizational support
Type of inner source software	Discrete software packages, such as utilities (e.g., XML parser), compilers, shells	Critical asset that plays an important role for the organization

receive such resources, in terms of budgets and person-years. Case studies of the project-specific model have been reported for Philips Healthcare (Wesselius 2008) and Alcatel-Lucent (Gurbani et al. 2006, 2010).

14.2.3.3 Comparison of Inner Source Models

Stol et al. (2011) presented a comparison of the two Inner Source models, summarized in Table 14.1. This comparison is based on observations from the current literature on Inner Source, and as such, these characteristics are not prescriptive. Organizations may have Inner Source projects that have characteristics of both models.

In the infrastructure model, all four characteristics in Table 14.1 are rather optional and casual, whereas for the project-specific model they are strategic and critical. For example, reuse in the infrastructure-based model is opportunistic and ad hoc, whereas in a project-specific model, reuse is a strategic goal. As a result, support is essential in the project-specific model, and typically one can observe a special organizational unit (i.e., a core team) that takes responsibility of the shared asset that is managed as an Inner Source project.

14.2.4 Inner Source Project Management

Open Source style development is often misinterpreted as a "chaotic" or "unmanaged" approach to software development, while it is better characterized as "self-organizing." Common Open Source governance models range from highly centralized, typically led by a "benevolent dictator for life" such as the Linux Kernel, to decentralized ones, such as GNU Linux. In between those two mentioned models sits a council-like governance model that is used, for example, in the

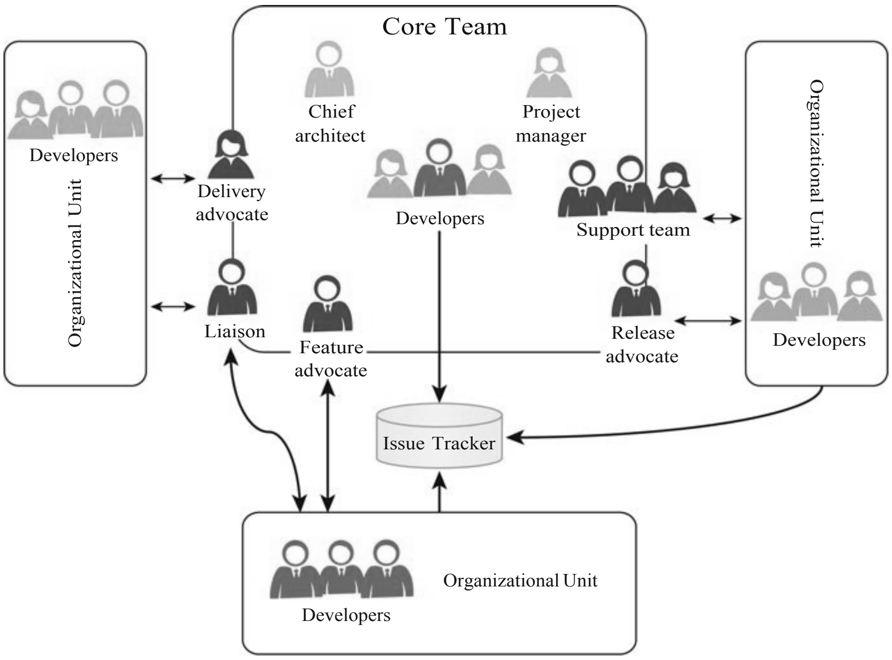

Fig. 14.2 Roles of an Inner Source core team (based on Gurbani et al. 2010)

Apache project. Hence, just as successful Open Source projects can have an extensive "management layer," so too do Inner source projects need management.

Little is written on project management in Inner Source projects. A notable exception is a study by Gurbani et al. (2010), which identifies a number of roles that emerged in Alcatel-Lucent's Inner Source project. The diagram in Fig. 14.2 presents our analysis of their presentation of these roles. In the remainder of this chapter, we assume that an organization consists of one or more organizational units, such as departments, teams, or business divisions.

It is important to emphasize that these roles emerged within the core team at one organization and are not prescriptive for an Inner Source initiative. They do, however, provide a useful reference model for organizational roles in Inner Source. We describe these roles briefly.

A liaison has the overall responsibility for the shared asset and oversees the activities performed by the core team. The liaison also interacts with the organization's internal customers (i.e., different organizational units) about, for example, feature requests. The liaison works closely with the shared asset's chief architect; the chief architect can be compared to a typical OSS project's benevolent dictator, who has strong technical skills and whose key responsibility is strategic road mapping and maintaining the shared asset's architectural integrity.

A support team (Gurbani et al. (2010) refer to them as construction, verification and load bring-up engineers) provides operational support to the business units.

This includes release management tasks, verification, documentation, and writing release notes. A project manager has overall project management responsibility, which includes release planning, project monitoring, and process compliance. A release advocate takes responsibility for a specific release and interacts closely with the organizational units so as to assess the potential impact of the release on the organizational units' products that integrate the shared asset. A delivery advocate is assigned to an organizational unit that becomes a new customer of the shared asset, so as to assist in integrating the component into a product. Gurbani et al. (2010) pointed out that an organizational unit might have specific tools, which complicates the task of integrating the shared asset. The delivery advocate also assists in ensuring that contributions from the organizational unit fit within the shared asset's overall architecture. Finally, a feature advocate is responsible for seeing a particular feature to completion.

14.3 A Framework for Understanding Project Management in Inner Source

Inner Source and project management are both complex and multifaceted topics with wide scopes and comprising many different aspects. One goal of this chapter is to identify key aspects of project management that are affected by adopting Inner Source. In order to define the focus of our study, we defined a framework based on a number of aspects that we consider important for understanding project management in Inner Source. The remainder of this chapter uses the framework to structure the results of our analysis. We derived this framework from a number of sources that have addressed the topic of project management in detail, which we briefly outline below.

Much has been written on project management in general, but also in a software engineering context as exemplified in this book. An important source is the Project Management Body of Knowledge (PMBOK) (Duncan 2013). This is a general guide to project management without a specific focus on software development. It includes an extensive range of factors, or topics, which are listed in the first column of Table 14.2. However, not all topics are related to Inner Source, which is why PMBOK is not an optimal instrument to focus our study.

Besides the project management field, the software engineering field also has a "body of knowledge," called SWEBOK (Abran and Moore 2004). SWEBOK is an ongoing project, currently in its third version, and "accepted knowledge" of the software engineering field. Similar to PMBOK, SWEBOK defines a number of themes, listed in the second column of Table 14.2. While all are specific to software engineering, not all themes are relevant to project management. Hence, SWEBOK would not be an optimal choice as a framework either.

In addition to these two potential sources, various books have been written on software project management (SPM), such as that by Hughes and Cotterell (2009).

Table 14.2 Themes discussed in various sources relating to project management

PMBOK	SWEBOK	SPM textbook (Hughes and Cotterell 2009)
– Integration management – Scope management – Time management – Cost management – Quality management – Human resource management – Communications management – Risk management – Procurement management – Stakeholder management	– Initiation and scope definition – Software project planning – Software project enactment – Review and evaluation – Closure – Software engineering measurement	– Project evaluation and program management – Selection of appropriate project approach – Software effort estimation – Activity planning – Risk management – Resource allocation – Monitoring and control – Managing contracts – Managing people in software environments – Working in teams – Software quality

This source includes all factors from both PMBOK and the "Software Engineering Management" part of SWEBOK; the third column in Table 14.2 lists the themes discussed by Hughes and Cotterell. However, while all themes identified by Hughes and Cotterell are relevant to project management and software engineering, not all are relevant to Inner Source. In order to define a sound framework, we identified four key themes that encompass the relevant topics from the three sources listed in Table 14.2. These are

1. **Process management**: This includes deciding in detail what development process to use during the project. In many cases, this is done by tailoring an organization-level process model to a project-specific process model that fits the constraints and requirements of a specific project.
2. **Project planning**: This includes traditional planning activities such as activity planning, effort estimation, and resource allocation.
3. **Monitoring and taking actions**: This includes activities for monitoring and control, as well as activities related to risk management.
4. **Human issues**: This includes issues related to human resources, people man- agement, and activities that support a healthy team climate.

These themes are used to structure the remainder of this chapter. We describe the themes in further detail below. This section ends with an overview of a number of key concerns of project management in Inner Source that require attention.

14.3.1 Process Management

Process management is concerned with selecting, tailoring, and aligning a suitable software development process for a project. That is, it is a "high level" and often organization-wide project management activity. Since the use of heterogeneous processes may also be an issue in conventional software development contexts, one could argue there is little difference with an Inner Source approach. There is no single "conventional process" and no single Open Source process that is used in Inner Source. Therefore, there is no single way of, for example, adopting and tailoring processes in any of the cases. Each Open Source project has its own set of practices and customs that have emerged over time. However, as outlined in Chap. 13, there are a number of common characteristics. Likewise, an Inner Source initiative is also tailored to an individual organization. Section 14.2.3 discusses the two major Inner Source adoption models, which is the first major difference between Inner Source initiatives. Furthermore, each initiative is shaped by the context and constraints of the organization. For example, Philips Healthcare develops a Software Product Line (SPL) for their medical equipment, whereby the SPL platform is managed as an Inner Source project (van der Linden 2009). As such, the company is subject to regulations set forth by the Food and Drug Administration (FDA) in the United States. One implication of this is that the process needs to be traceable, and regulatory bodies (such as the FDA) conduct audits regularly to inspect the process.

Even though there are many different development methods, making it difficult to identify common characteristics, we discuss a number of common questions that may arise in relation to process management.

Two types of processes are of interest in this respect. First, there is the process that is selected for development of an Inner Source project. Second, there are the processes that are used by the customer teams that wish to integrate (or use) the Inner Source project, and possibly contribute to it. This potentially large variety in customer projects means that there may be a range of different processes that interact with an Inner Source project's process. Some customer projects may follow a strictly stage-gated model (e.g., waterfall), whereas others follow more iterative approaches such as Agile methods (e.g., Scrum). When customer projects wish to contribute to the shared asset, they must consider the alignment of their own process and the process used by the core team that develops the shared asset. A misalignment of such processes may result in problems when the shared asset is integrated, or worse, when a customer team misses a product release deadline (Stol et al. 2011).

There are also requirements regarding the process for an Inner Source project. This process must be formal and sufficiently rigorous to fulfill the requirements of a product and its evolution, that is, handle contributions from a variety of sources in an efficient way. It must also be aligned to other processes at the organizational level, which are not necessarily adapted to this way of working. That is, the process must resemble an Open Source process with its mechanisms for evaluating contributions, widely available and accessible information, selecting among candidate changes in a timely manner, etc., and at the same time adhere to the typical organizational level process requirements that are characterized by milestones and deadlines. Conflicts may arise between a general organizational process and an Inner Source process (Riehle et al. 2009).

Customers of an Inner Source project may use a variety of processes, which is why it is difficult to outline general guidelines as to how these should be aligned. In one case study, Lindman et al. (2008) found that they are typically agile. However, one type of required process tailoring relates to contributions to an Inner Source project. Contributors must adhere to the requirements of an Inner Source process prescribed by the core team, and the level of formality enforced by the core team must be taken into consideration.

14.3.2 Project Planning

Project planning includes traditional activities such as activity planning, effort estimation, and resource allocation, but also more general activities and tasks such as coordination of development activities and prioritization of different implementation alternatives.

There are many planning activities of a more general nature that are important for Inner Source management. For example, Gurbani et al. (2006) emphasized that the core team must have a long-term vision of the evolving shared asset. Since many projects may become dependent on the shared asset and future changes, it is important to be able to foresee the need of different projects and to be able to communicate the project vision.

It is also important to understand that contributing projects may not be as focused on general solutions as the core team must be. It is natural that contributing projects are more focused on the specific development of their project and consider the Inner Source product as only one part of their project. This means that a core team must coordinate the development schedules from different contributing teams, and as much as possible, minimize the risk of duplicate work. This may involve in some cases the prioritization and to some extent negotiation of requirements from different projects. Long-term planning also includes evolution planning and distinguishing general development from customer-specific changes (Gurbani et al. 2006). Gurbani et al. (2010) suggested that this required a "full time project manager." Product evolution planning can be supported by keeping a list of candidate changes, and trying to find common themes among future changes.

14.3.3 Monitoring and Taking Actions

This part of project management describes how managers can follow up work and progress, and based on that take actions with the objective to correct for deviations between plans and expectations, and actual progress.

As in the previous section, this text focuses on the monitoring carried out by an Inner Source coordinator, i.e., core team, and not on the monitoring carried out by the contributing projects.

One important type of monitoring is that of quality assurance of contributions. An Inner Source coordinator may carry out this quality assurance when code is contributed, but this may prove to be a bottleneck (Gurbani et al. 2006). One suggested benefit of Inner Source is to have “truly independent” peer review as an effect from a large community who have access to the code (Linus’s Law).

An Inner Source coordinator can also track features during development and compare to the vision of the evolving code. If there are differences, which in some way mean that future contributions will not be accepted, they can be identified as soon as possible.

14.3.4 Human Issues

The fourth and last theme of our framework is that of human issues and is based on some of the observations that have been reported in the Inner Source literature. One aspect of human issues that has been addressed by Gurbani et al. (2006) concerns the fact that different contributing projects may have processes that are different from the way that contributions are managed by the core team. This includes, for example, how contributions are inspected and how quality is assured.

An increased transparency of the development process that is necessary to facilitate Inner Source may introduce friction (Gaughan et al. 2009; Melian and Mähring 2008). Developers may experience an increased openness of the process as a “fish-bowl,” and an increased pressure on performance. They may also see the openness as a threat to their unique competence and skill set since more people now may work on the same code with the same type of problem. It may require staff to develop new skills with respect to communication and interaction. These aspects may not be a problematic in all Inner Source initiatives, but cognizance of such potential issues may help address such issues before they arise and escalate.

There are also some positive aspects of this nature. Other developers may experience an open environment as very positive and rewarding. Others may see the open environment and the possibility to voluntarily contribute as a means for professional improvement or as a way to demonstrate their expertise, and thereby rewarding. Such rewards may positively affect developers’ motivation, a topic that is discussed in more detail in Chap. 10.

14.3.5 Summary

Table 14.3 summarizes some of the tension points related to the four project management themes presented above, that may arise as a result of introducing Inner Source. For example, if there is a focus on applying an available process in conventional development, a difference in Inner Source is that there are now a number of processes in organizational units that must be aligned. In conventional

Table 14.3 Comparison of traditional project management and potential tension points in Inner Source

Element	Traditional project management	Key difference in inner-source
Process management	Enforce common processes Top-level coordination enforcement.	Alignment of different processes may be challenging
Project planning	Predefined, well-organized, use of planning tools (e.g., Gantt charts) to "predict" delivery. Project costs more easily predicted and calculated (Gaughan et al. 2007)	Planning of "chaotic" OSS style project may be unnerving. Costing of Inner Source shared asset is more difficult
Monitoring and taking actions	Standard methods for following up progress of projects. Traditional punish/reward model	Follow up of more high-level attributes like plan for future releases
Human issues	Option to deliver "good enough" code as it is not visible to most people Typically no encouragement of initiatives to contribute (maybe voluntary) to work outside own project Hierarchical business organization (Gaughan et al. 2007)	Contributions or code may be in an "embarrassing" state (Dinkelacker et al. 2002) Conflicts OSS style (ego) strong personalities; "bullying" by technological experts who take ownership Opening up code may face objections from developers who fear losing their job Employees may resist change (Gaughan et al. 2007)

development, there are a number of well-known methods for project planning, while in Inner Source there is a need to plan and synchronize different initiatives from organizational units that will encourage contributions. This also means that monitoring must concern several organizational units, for example, with respect to what each will contribute and how this is aligned to the shared asset as a whole. Finally, a number of human issues need to be considered as well.

14.4 Case Studies

In this section, two industry case studies are presented in order to illustrate project management issues in Inner Source. Organizations that wish to adopt Inner Source need to understand their current project management approach so that they can assess the extent to which this aligns with an Open Source development approach. We conducted a case study at one organization that had a strong interest in adopting Inner Source, to illustrate such an assessment of Inner Source alignment prior to adoption. This is presented in Sect. 14.4.1.

Whereas the first case study presents potential project management tension points that may arise prior to, or during adoption of Inner Source, we also conducted a case study at an organization with an established track record in Inner Source adoption. This second case study sheds light on a number of actual challenges in

Inner Source project management and how those issues have been addressed. This case study is presented in Sect. 14.4.2.

14.4.1 A Case Study of Inner Source Alignment

The first case study was conducted in a multinational software and hardware company, based in Sweden (hereafter referred to as “ToolSoft”). ToolSoft has been using Open Source software in its products and has been involved with large Open Source communities. The goal of the case study was to understand the alignment of ToolSoft’s development practices with the Open Source development practices. Therefore, a first step was to identify a set of software development practices that are typical for Open Source development and compare them to the current development practices in the company. In order to do this, practices typical for Open Source development were identified by analyzing the most important aspects of the Open Source projects hosted under the Apache project. The identified practices were also validated by studying Fogel (2005) on how to run a successful Open Source project. The identified practices are listed in Table 14.4. After the comparison was made, the data was interpreted in the light of the aspects of the framework presented in Sect. 14.3, according to which the results are also presented in this chapter.

Traditional development practices are closely related to conventional software project management (SPM) discussed in more detail in Sect. 14.3. Traditional SPM is analyzed through its main features: process management, project planning, monitoring and taking actions, and human issues. In order to understand how Open Source development practices identified in this case study can be applied within a conventional SPM context, the four features of the traditional SPM are further examined through the framework presented in Table 14.4, which outlines a number of aspects specific to Open Source development.

14.4.1.1 Process Management

The process management aspect of software project management is closely related to the Open Source infrastructure aspect. An infrastructure portal (or “forge”) hosts information about Open Source development practices including a community guide and information on source code repositories, development roadmap, etc. All individual projects developed by a community need to comply with the infrastructure aspects, just like any specific project developed within a more traditional environment needs to comply with process management defined on the organization level.

In this study, we found that ToolSoft’s development practices are mostly aligned with Open Source practices with respect to the infrastructure aspects. However, some of the content hosted under the infrastructure portal was found to be

Table 14.4 Open Source development practices considered in analyzing ToolSoft's software development practices

Aspect	Category	Element
Infrastructure	Product info	Features, documentation, FAQ, news road map, security
	Code access	Download location, binary package, release notes
	Community guide	Community overview, community roles, coding conventions, commit conventions, building and testing, debugging, mailing lists, bugs/issues, releases
Communication	Standardized	Message, channel, norm
Management	Meritocracy	Role, promotion, authority

incomplete and out of date. In an Open Source setting, up-to-date information hosted under the portal is crucial for understanding how a project operates, for example, what the project goals are, current issues and bug tracking, rules of conduct, developers guide, and documentation. In a traditional setting, much of this information is disseminated through different channels, either through interpersonal communication, unarchived electronic communication, or a project's specific documentation. What can happen in practice, is that different projects working on the same parts of software do not synchronize documentation, or a task of completing documentation is not given a sufficient priority, and resulting in outdated documentation.

14.4.1.2 Project Planning

The project planning aspect of traditional software project management deals with identification of activity, effort estimation, and resources allocation. In an Open Source community, an activity or task is identified through a transparent communication process, for example, a community participant identifies a new feature to be implemented or reports a bug through an open online forum or email list. The proposed activity is then further discussed and assessed among community participants. Any community participant can decide to work on a task. Hence, there may be multiple community participants working on the same activity. This is a very different approach to resource allocation from a traditional planning approach, where a resource is assigned to an activity, rather than a resource being able to assign him/her self to the activity. Even though effort estimates are normally provided for activities in an Open Source realm, a time constraint of an activity is not normally enforced in the same sense as in a more traditional closed source environment.

14.4.1.3 Monitoring and Taking Actions

The transparent communication aspect of Open Source development ensures that many individuals look at the way a product is being developed, and thus, the "many eyeballs” effect (Linus’s Law) often found in Open Source can be related to the monitoring and taking actions aspect of the traditional software project management. In Open Source development, the monitoring aspect is steered through a transparent process where participants provide feedback on actions taken.

In this case study, we found that the communication characteristics observed in ToolSoft have a high level of misalignment with Open Source practices. As employees were seated closely to each other within the office plan, people tended to favor informal meetings and discussions over electronic communication. This way, the communication process is not transparent, or traceable, and may have to be repeated by different community members. Open source communities use electronic forms of communication that facilitate transparent and traceable discussions. Transparent discussions enable relevant participants and decision makers to get involved, which may help in resolving issue in a timely fashion. Archived discussions can be referenced in order to understand why certain decisions were taken in the past or to find out how similar issues were resolved. Implementing transparent and archived communication archives could improve efficiency in ToolSoft by ensuring that relevant resources are involved on time by creating searchable problem/resolution archives and by decreasing the amount of time spent in repetitive and less efficient ways of communication.

14.4.1.4 Human Issues

The transparent nature of communication in Open Source development helps in understanding how the project functions, the importance of contributors’ roles, and thus, ensures that the participant roles are assigned in a meritocratic way. A community guide defines the rules of engagement within the community, especially in terms of online communication norms. Open Source communities have recognized the importance of friendly, standardized, and efficient communication in overall project success (Fogel 2005). Hence, the community guide and open communication aspects are related to the human issues feature of software project management.

Our assessment of the management aspect showed alignment of the defined roles in ToolSoft and Open Source development practices, but in practice the roles in ToolSoft exhibit overlapping characteristics. For example, we found that software architects sometimes took on responsibilities of project leads. Such overlapping and conflicting roles can decrease the overall process and product efficiency. One example of this is sacrificing some aspect of a product’s technical maturity to meet a deadline.

14.4.1.5 Conclusion

In this case study, we found that implementing some characteristics of typical Open Source practices while retaining a more traditional software development approach in terms of communication and project management may hinder the potential efficiency of Inner Source. In order to better understand the actual effects of a transition to Inner Source and Open Source practices, more research is needed. This would increase understanding of benefits and drawbacks of Inner Source in an organization, and potentially offer ways to tailor practices to closed development environment with a limited number of resources and projects having a time constraint as a critical attribute.

14.4.2 A Case Study of Project-Specific Inner Source

The second case study was conducted at an organization that we refer to as "GlobalSoft." GlobalSoft is a multinational organization in a regulated domain. GlobalSoft has adopted a number of OSS development practices and augmented their conventional mechanism for project management. Some of this case study's findings were previously reported by Stol et al. (2011); the results presented in this section focus in particular on project management issues, taking into consideration the four themes introduced in Sect. 14.3.

The GlobalSoft organization consists of a number of business units. Each business unit is specialized in a specific domain for which they develop products. Each business unit has therefore highly specialized expertise and in-depth knowledge of these technologies.

Figure 14.3 shows a representation of the key elements of the Inner Source initiative at GlobalSoft. The figure shows that a core team develops and releases versions of the shared asset. The core team consists of architects and developers, as well as a support team (similar to what is depicted in Fig. 14.2). The core team can work closely with business units in so-called collaborative development projects, to develop a new (or enrich an existing) component, which is then integrated back into the shared asset. Effectively, these collaborative development projects include a feature advocate from the business unit. A steering committee defines a roadmap and decides which features are implemented. Business units integrate, and possibly customize the shared asset. As new companies are acquired, they become part of the organization as new business units, and any software that is brought in is scrutinized for potential reuse.

While Inner Source offers many potential benefits (as discussed in Sect. 14.2), it may also introduce new challenges, which are sometimes similar to those associated with using OSS in product development (Stol et al. 2011). The remainder of this section presents a number of challenges related to project management and how they were addressed by the GlobalSoft.

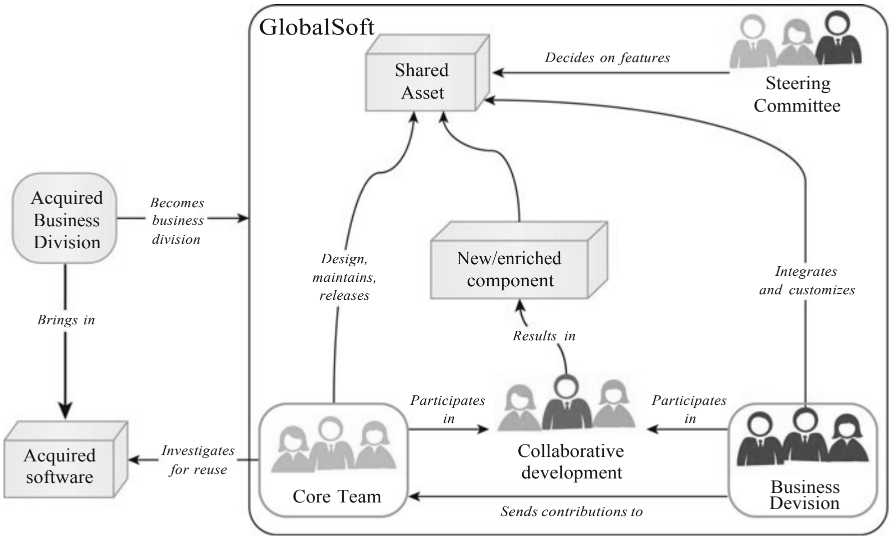

Fig. 14.3 Conceptual model of Inner Source in the GlobalSoft organization (adapted from Stol et al. 2011)

14.4.2.1 Process Management

A key concern in process management was finding agreement on the software development processes to be used. In the last two decades, GlobalSoft acquired many companies in the same domain, which have become business units. Besides each organization's unique culture, an organization may also have a set of established development processes to which they are accustomed.

One challenge that this study revealed was a misalignment of the software development life cycles at the business units and the core team. Business units want to focus exclusively on differentiating, value-adding functionality, and prefer that common functionality shared by different products be implemented by the core team. To that end, business units could send their requirements to the core team, which would then implement a certain module or component. However, the integration tests were performed by the requesting business unit; the core team did not build actual products and as a result did not do any integration tests, and a such did not find any problems in using the new functionality. By the time a business unit was ready to integrate the new component and identify problems (e.g., missing functionality or lack of attention for quality attributes such as performance), the core team had focused their attention to new tasks. As a result, little support in terms of defect fixing could be expected, which was a burden for the business unit that was trying to deliver a product to the market on time.

In order to prevent such problems, GlobalSoft adopted the concept of so-called collaborative development projects. This can be considered a hybrid solution of "pure" OSS style defect fixing by noncore developers, but with close involvement of the core team. In practice, this resulted in temporary, virtual teams that work together on either a new component or to enrich an existing component.

14.4.2.2 Project Planning

As outlined in Sect. 14.3, project planning relates to activities such as activity planning, effort estimation, resource allocation, but also includes feature prioritization. Many of these topics seem nonexistent in OSS projects, since in Open Source developers self-select tasks that they feel they can do, or they feel need to be done. This self-organizing feature of OSS style development is more difficult to combine in commercial software development, where schedules need to be met and budgets need to be respected.

At GlobalSoft, a steering committee developed a roadmap that outlined the future development plans of the shared asset. The steering committee had representatives from the business divisions as well as from the core team. This way, input was received from both the supplier side as well as from the customer side, which ensured that (1) the core team better understood what was needed by the business divisions (i.e., the shared asset's users) and (2) the business divisions could align their own roadmap of future product evolution with the shared asset on which they based their product.

One issue we found in this case study is that the development capacity of the core team was a bottleneck. This was due to the fact that the core team developed a platform that was used by many different business units, each with many feature requests. This bottleneck was in fact one of the motivations to adopt OSS development principles. Business divisions can—at their own cost—request development of certain features, but this "solution" is rather limited, given the restricted capacity of the core team.

14.4.2.3 Monitoring and Taking Actions

The case study organization used commonly used infrastructure offered by Collab. Net.[6] This includes common infrastructure such as a mailing list, a wiki, and an issue tracker, which facilitated knowledge sharing, community interaction, and tracking of issues. As developers encountered difficulties, the mailing list provided a means to ask questions and share knowledge. Architects of the core team monitored these lists regularly and provide feedback where possible.

As outlined above, the organization had grown significantly over time, as companies were acquired that became new business units. As a result of this, the scope of the shared asset (the product line platform used as a foundation for most products) was evolving. Instead of a static scope (with a fixed set of features and use

[6] www.collab.net

cases), the shared asset's scope was dynamic. As the new business units became new customers of the shared asset, new use cases had to be considered, so that the new business unit could also benefit from the platform. This could either be done by porting the existing product to GlobalSoft's platform or to rebuild the new business unit's software using the platform as a foundation.

14.4.2.4 Human Issues

There are a number of human issues to consider in Inner Source project management. A key challenge in building an internal, organization-wide and interactive community is getting developers to contribute. Without active members who answer questions of other people in the community, an Inner Source initiative may not become very successful.

A key factor that should be considered is building an environment in which all parties involved become supportive of the Inner Source program (Stol et al. 2014). In particular, incentives should be clear to each development group so that any potential tension between the core team (i.e., supplier) and the business divisions (i.e., customers) is prevented. Business divisions need to see clear benefits from actively participating in the community in order to prevent a "them versus us" atmosphere. This involves clearly communicating—and demonstrating—potential benefits. Some business divisions were more receptive of GlobalSoft's Inner Source initiative than others.

Similar to what can be observed in Open Source communities, we found that typically there was an evolution—or a "learning curve"—in involvement in GlobalSoft's Inner Source community. Usually this started with using some components of the product line's platform (similar to an OSS product's users). As more parts of a product's application were migrated to the new platform (i.e., the shared asset at GlobalSoft), a business unit would increase its interaction with the core team. At some point, a business unit could decide to use components that are in development (rather than a released snapshot version) to benefit from the latest available functionality, and a business unit could want to contribute certain functionality that they required (comparable to OSS users (or "lurkers," even) becoming contributors). In GlobalSoft, this process worked better for some business units than for others. In particular, business divisions that actively followed the core team closely in their development had significantly fewer problems than those who treated the core team as a traditional "black box" software supplier.

As outlined above, GlobalSoft acquired a large number of other companies over time, each of which brought in their own software and systems. In many cases, these systems would have a different architecture than GlobalSoft's product line. One of our findings is that GlobalSoft took a conscious approach to sit together with the people who designed and developed the newly acquired systems. It was felt important to come to an understanding about the design rationale for the different systems, and to find agreement on how a new business division's software could make use of the shared asset. Respect for the acquired company's culture is important in this context, rather than prescribing and requiring conformance to GlobalSoft's existing architecture. It was felt that a close collaboration, understanding of each other's software assets, and identifying how to let the different systems grow toward each other would yield much better results (in both the technical aspect as well as in "goodwill") than enforcing GlobalSoft's architecture.

14.5 Discussion and Future Work

This section continues the discussion on differences between conventional development and Inner Source development. In particular, the objective is to discuss how introducing Inner Source affects software project management. It is important to understand that each Inner Source initiative must be tailored to the context of an organization so as to consider the various constraints an organization may have to address (e.g., regulated environments) as well as the organizational culture.

The decision to introduce a strategy such as Inner Source is always based on a trade-off. On one hand, there are a number of perceived advantages such as increased reuse and transparent communication. However, challenges may arise as well. A key issue is to recognize that the different organizational units that work with an Inner source core team are likely to use different development processes, which can result in significant coordination challenges. The case study of GlobalSoft illustrated how such process-misalignment manifested. To overcome such issues, new coordination mechanisms can be required, or existing organizational governance mechanisms can be adjusted so as to better plan development of a shared asset, and facilitate synchronization of development processes—in the case of GlobalSoft, a new collaborative development mechanism emerged.

Challenges may also arise on the operational level for developers, who work in the various organizational units and who contribute to development of a shared asset. An increased transparency of the process in general and the visibility of the contributed code may also result in tension points. For example, whereas code contributions used to be limited in visibility to a developer's project team members, all project artifacts can now be scrutinized by an organization's global developer community. Some people may also see the electronic communication that is typically used in an Open Source environment as over-formalized. This would also result in a fully transparent (and archived) communication throughout an organization. This may mean that some people would prefer to have informal meetings instead of using, for example, mailing lists.

One strategy to address several of the challenges is to start with a code asset that is already used by several projects, and then gradually introducing the concept for other and new assets. We argue that it is an advantage to start with a part of the code and the organization where the change is seen as positive, that is, in line with creating a "short-term win" (Kotter 1996).

The key observations from the case studies are summarized in Table 14.5. The results in the table suggest a few steps to take in transforming from a conventional

Table 14.5 Key findings of the two case studies

Theme	ToolSoft	GlobalSoft
Process management	Project portal that facilitates "self-management" may exist but may not be used fully	– Collaborative development projects to overcome process misalignment
Project management	Higher degree of engagement of all project participants through transparent communication process to deliver high-quality software products up to users' specifications can be expected	– Steering committee is useful to gather organization-wide input and synchronize efforts – Extra "purchasing" of critically needed software development possible but limited to capacity of core team
Monitoring and taking actions	Establish transparent and archived communication to identify and resolve issues more efficiently	– Dynamic scope of shared asset due to new required use cases
Human issues	Facilitate friendly communication atmosphere and to promote based on merit to build a healthy organization culture needed	– Need clear incentives so as to engage people – Learning curve may be steep due to novelty of approach – Respect for developers and organizations' cultures is key to successful collaboration

development approach to Inner Source. For example, an infrastructure for managing information needs to be used more extensively, and process synchronization mechanisms such as collaborative development projects can be adopted to overcome process misalignment issues.

How to carry out the introduction of Inner Source will vary across organizations and heavily depend on contextual factors. We argue that it is important to identify what gains are considered important and focus on those one at a time. It is also important to identify what challenges may arise and to prepare mitigating actions to overcome them.

Concerning process management, an important part of Inner Source management is to align different processes in different organizational units. This is also reflected in the importance of monitoring of different development initiatives in different organizational units. Conventional project management is still taking place in contributing organizational units, while an important part of Inner Source management concerns coordination of initiatives. All management levels in an organization should also be aware of the potential human issues that can affect the introduction of Inner Source. That is, introducing Inner Source will mean a number of changes at the tactical and operational levels when it comes to synchronizing development activities in different business units in the organization. This is also a basis for increasing reuse between business units, which by many is seen as an important goal of introducing Inner Source.

14.5.1 Future Work

As mentioned in this chapter, Inner Source is an emerging approach to software development. Although the first studies on this topic were published in the early 2000s, the field is still in its nascent phase, and more research is necessary to better understand how benefits (outlined in Sect. 14.2) can be achieved. We conclude this chapter by outlining a number of directions for future work.

- While different Inner Source adoption models exist (Gurbani et al. 2010), further studies of how organizations embrace Open Source development practices will be a welcome addition to the literature.
- While there exist a few reports of how Inner Source supports the development of a software product line (Van der Linden 2009), there are a few studies of reuse of Inner Source components. Such studies do exist in an Open Source context (e.g., Capiluppi et al. 2011), which can be used as a template to design studies of reuse of Inner Source components.
- In this chapter, Inner Source has been compared to traditional project management aspects. Further research can include comparison of Inner source and management in Agile projects.
- Quantitative studies to better understand how certain benefits can be achieved. Such studies typically identify dependent and independent variables, and can identify the relationship between those variables so as to be able to "predict" how a certain benefit can be achieved.

Acknowledgments The authors wish to thank Brian Fitzgerald for useful feedback on an earlier draft of this chapter. This work was conducted within the ITEA2-SCALARE project, supported by Vinnova and Enterprise Ireland, and further by Science Foundation Ireland grant 10/CE/ I1855 to Lero — the Irish Software Engineering Research Centre (http://www.lero.ie), and by the Industrial Excellence Center EASE—Embedded Applications Software Engineering (http://ease. cs.lth.se).

References

Abran A, Moore JW (2004) Guide to the software engineering body of knowledge. IEEE
Ågerfalk PJ, Fitzgerald B (2008) Outsourcing to an unknown workforce: exploring open-sourcing as a global sourcing strategy. MIS Q 32(2):385–409
Capiluppi A, Stol K, Boldyreff C (2011) Software reuse in open source: a case study. Int J Open Source Softw Process 3(3):10–35
Capiluppi A, Stol K, Boldyreff C (2012) Exploring the Role of Commercial Stakeholders in Open Source Software Evolution. In: Hammouda I et al (eds) OSS 2012, IFIP AICT 378, pp 178–200
Dinkelacker J, Garg PK, Miller R, Nelson D (2002) Progressive open source, 24th international conference on software engineering (ICSE), Orlando, FL, pp 177–184
Duncan WR (2013) A guide to the project management body of knowledge (PMBOK®guide), 5th edn. Project Management Institute (PMI), Newtown Square
Fogel K (2005) Producing open source software: how to run a successful free software project. O'Reilly Media, Sebastopol

Gaughan G, Fitzgerald B, Morgan L, Shaikh M (2007) An examination of the use of inner source in multinational corporations: a preliminary framework to understand inner source software development. In: Proceedings 1st OPAALS conference, pp 48–60

Gaughan G, Fitzgerald B, Shaikh M (2009) An examination of the use of Open Source software processes as a global software development solution for commercial software engineering. In: 35th Euromicro conference on software engineering advanced applications (SEAA), pp 20–27

Gurbani VK, Garvert A, Herbsleb JD (2006) A case study of a corporate open source development model. In: 28th international conference on software engineering, pp 472–481

Gurbani VK, Garvert A, Herbsleb JD (2010) Managing a corporate open source software asset. Commun ACM 53(2):155–159

Hughes B, Cotterell M (2009) Software project management. McGraw-Hill, New Delhi

Ierusalimschy R (2008) Lua Mailing List, reply of Roberto Ierusalimschy, one of the developers of Lua, Friday, 27 June. http://lua-users.org/lists/lua-l/2008-06/msg00407.html

Kotter J (1996) Leading change. Harvard Business Review Press, Boston

Lindman J, Rossi M, Marttiin P (2008) Applying open source development practices inside a company. In: Russo B, Damiani E, Hissam S, Lundell B, Succi G (eds) Open source development, communities and quality. Springer, New York

Lindman J, Riepula M, Rossi M, Marttiin P (2013) Open source technology in intra-organisational software development–private markets or local libraries. In: Ericsson Lundstrom J, Wiberg M, Hrastinski S, Edenius M, Ågerfalk PJ (eds) Managing open innovation technologies. Springer, Berlin

Melian C, Mähring M (2008) Lost and gained in translation: adoption of open source software development at Hewlett-Packard. In: Russo B, Damiani E, Hissam S, Lundell B, Succi G (eds) Open source development, communities and quality. Springer, New York

Morgan L, Feller J, Finnegan P (2011) Exploring inner source as a form of intra-organisational open innovation. In: Proceedings European conference on information systems

Oručević-Alagić A, Höst M (2010) A case study on the transformation from proprietary to open source software. In: Boldyreff C, González-Barahona JM, Madey GR, Noll J, Ågerfalk PJ (eds) Open source software: new horizons. Springer, Boston

Riehle D, Ellenberger J, Menahem T, Mikhailovski B, Natchetoi Y, Naveh B, Odenwald T (2009) Open collaboration within corporations using software forges. IEEE Softw 26(2):52–58

Stol K, Babar MA (2010) Challenges in using open source software in product development: a review of the literature. 3rd workshop on emerging trends in FLOSS research and development, co-located with international conference on software engineering, pp 17–22

Stol K, Babar MA, Avgeriou P, Fitzgerald B (2011) A comparative study of challenges in integrating open source software and inner source software. Inf Softw Technol 53(12):1319–1336

Stol K, Avgeriou P, Babar MA, Lucas Y, Fitzgerald B (2014) Key factors for adopting inner source. ACM Trans Softw Eng Methodol 23(2)

Van der Linden F (2009) Applying open source software principles in product lines. Upgrade 10 (2):32–41

Van der Linden F, Lundell B, Marttiin P (2009) Commodification of industrial software: the case for open source. IEEE Softw 26(4):77–83

Vitharana P, King J, Chapman HS (2010) Impact of internal open source development on reuse: participatory reuse in action. J Manage Inf Syst 27(2):277–304

Wesselius J (2008) The bazaar inside the cathedral: business models for internal markets. IEEE Softw 25(3):60–66

Biography Martin Höst is a Professor in Software Engineering at Lund University, Sweden. He received an M.Sc. degree from Lund University in 1992 and a Ph. D. degree in Software Engineering from the same university in 1999. His main

research interests include software process improvement, software quality, risk analysis, and empirical software engineering.

Klaas-Jan Stol is a researcher with Lero—the Irish Software Engineering Research Centre. He holds a PhD in software engineering from the University of Limerick. His research interests include contemporary software development methods and strategies, including Inner Source, Open Source, crowdsourcing, and agile and lean methods, as well as research methodology and theory building in software engineering. In a previous role, he was a contributor to an Open Source project.

Alma Oručević-Alagić is a Ph.D. student at Lund University, Sweden. She received an M.Sc. degree in Software Engineering from the University of St. Thomas, St. Paul, Minnesota in 2002, and a Technical Licentiate degree in 2013 from Lund University. Her research interests include Open Source, Inner Source, and Network Analysis of software development communities.