On infrastructure for facilitation of inner source in small development teams

Linåker, Johan; Krantz, Maria; Höst, Martin

Published in:
Product-Focused Software Process Improvement

DOI:
10.1007/978-3-319-13835-0_11

2014

On infrastructure for facilitation of inner source in small development teams

Johan Linåker, Maria Krantz, Martin Höst

Software Engineering Research Group, Computer Science, Lund University
{johan.linaker,martin.host}@cs.lth.se,

Abstract. The phenomenon of adopting open source software development practices in a corporate environment is known by many names, one being inner source. The objective of this study is to investigate how an organization consisting of small development teams can benefit from adopting inner source and assess the level of applicability. The research has been conducted as a case study at a software development company. Data collection was carried out through interviews and a series of focus group meetings, and then analyzed by mapping it to an available framework. The analysis shows that the organization possesses potential, and also identified a number of challenges and benefits of special importance to the case company. To address these challenges, the case study synthesized the organizational and infrastructural needs of the organization in a requirements specification describing a technical infrastructure, also known as a software forge, with an adapted organizational context and work process.

Keywords: Inner source, Open source software, Software development practices, Software ecosystem, Life cycle, Programming teams, Software process models, Software reuse, Software forge

1 Introduction

Many open source software products have been successful in recent years, which have led to an increased interest from the industry to investigate how the development practices could be introduced in a corporate environment and take advantage of the benefits seen in open source projects. Such practices include e.g. universal access to project artefacts [8], early and frequent releases, and "community" peer-review [2].

Mistrik et al. [11] address how closed development organizations could benefit from open source practices as an area where further research is needed. Though studies conducted so far are quite limited, several success stories [1, 2, 8, 13, 20] can be found of large corporations adopting open source development.

The phenomenon of adopting these development practices in a corporate environment has in research been referred to by many names, e.g. *inner source* [17], *corporate open source* [2, 3] and *progressive open source* [1]. In this report we have chosen to use the term inner source, as described by Stol [17].

The changes required when adopting inner source in a corporate environment led Gurbani et al. [3] to suggest two different methods to effectively manage inner source assets; an infrastructure-based model and a project-based model.

In the infrastructure-based model, the corporation provides the critical infrastructure that allows interested developers to host individual software projects on the infrastructure, much like SourceForge[1] or Github[2] does with open source projects. Platforms like these, also known as software forges [13], can be resembled as component libraries where each project represents a component of different abstractions, e.g. modules, frameworks or executables. Developers can browse between the components and use or contribute to those they wish. The reuse of software can be considered opportunistic or ad hoc and there is no limitation on the number of projects to be shared within the organization. Success stories include cases from SAP [13], IBM [16, 19], HP [1, 10] and Nokia [7, 8].

In the project-based approach the software is managed in a project, instead of as a long-term infrastructure. Gurbani et al. [3] describe how an advanced technology group, or a research group funded by other business divisions in a corporation takes over a critical resource and makes it available across the organization. This team is often referred to as the "core team" and is responsible for the project and the decision making. Philips Healthcare [20] and Alcatel-Lucent [3] are two documented cases where this variant has been adopted.

In order to assess the applicability of inner source on an organization, Stol [17] developed a framework. This framework is based on reviewed literature and a case study of a software company referred to as "newCorp". Though the framework focuses on project-based models, it is based on success factors and guidelines described in both project-based and infrastructure-based case studies.The framework consists of 17 elements divided into four categories; Software product, Development practices, Tools and infrastructure, and Organization and community. These categories were inspired and mapped to those proposed by Gurbani et al. [2]. The elements can be found in the left column of Table 1. In a later publication Stol et al. [18] present an updated version of the framework where it has been restructured, now consisting of nine elements divided into three groups; Software product, Practices and Tools, and Organization and Community. The elements are more generalized and covers all of those described in the initial framework [17].

Adopting inner source requires significant effort and change management, which is one reason why it may be of interest to start on a smaller scale before investing globally. However, this requires an understanding of how inner source can be implemented on smaller teams and what parts that can be implemented and evaluated. This study is focused towards the latter and aims to contribute theoretically by using a framework by Stol [17] to assess the applicability of inner source, and based on the identified challenges synthesize a solution that addresses the organizations needs.

[1] `http://www.sourceforge.com/`
[2] `http://github.com`

The outline of this paper is as follows. In Section 2 the research methodology is presented and the results are presented in Section 3 and 4. The results are discussed and further analyzed in Section 5. The validity of the conducted research is presented in Section 6 and the research results are summarized in Section 7.

2 Methodology

This research is of a problem-solving nature and conducted as a case study with an exploratory strategy approach [14, 15]. The case company is experiencing problems in regards to its reuse of code and overall efficiency. The hypothesis is that the concept of inner source, as described earlier, can help the organization manage these issues (e.g. [1, 2, 8, 16]).

The organization was observed in order to further define the problem during spring 2012. Then it needed to be assessed whether inner source would fit the organization or not, and what the challenges would be. Based on the findings, the parts of inner source suitable for the organizations needs were synthesized in a requirement specification describing a technical infrastructure together with an organizational context and work process.

This improving approach can be compared to that of action research. However, this study has focused on the initial parts and proposed a solution. This is yet to be implemented and evaluated. I.e. the complete change process is yet to be observed. Due to limitations in time for the researchers and organizational conditions of the case company, this is left for future research.

2.1 The case company

An international software development firm, hereby known as "the case company", has been chosen for this study. The case company has a division based in a local office in Sweden which specializes in rapid software development and deployment of projects where the customers seek a combination of high quality and a fast release.

The scope is limited to the local division, though a network of corresponding divisions is established globally. The division of interest is divided into two teams with similar set-up and structure. Each team consists of 20-25 engineers including developers, testers and project managers.

2.2 Case study steps

The data collection and analysis was carried out as outlined in Figure 1, and described below.

2.3 Situation analysis

A situation analysis was conducted with the objective to describe and explain the current situation at the company. The main goal was to gain an understanding of how work is conducted within the organization and can be seen as an observational part of the research.

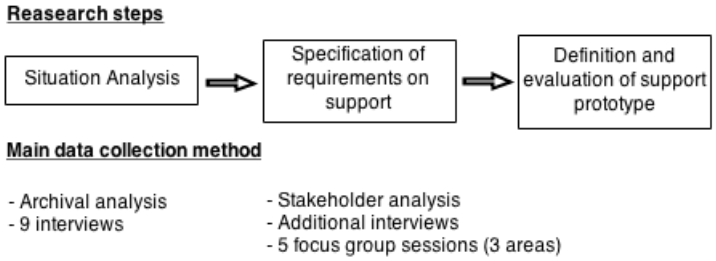

Fig. 1. Overview of data collection and analysis

Qualitative data was collected by studying documentation at the case company and by interviewing a sample group. The criteria for selecting people in this phase were that they should: be representatives from different areas, with different work tasks, to get a good variation of answers; have more than two years of work experience, within this or other companies; and be available for interviews.

In total 9 individuals were interviewed which included 4 project managers and senior back-end developers, 2 senior front-end developer, 1 junior front-end developer, 1 junior back-end developer, and 1 service manager.

The interviews were semi-structured with 20 questions prepared in advance. All interviews were recorded with both audio and supportive notes taken by the authors. The interview data was then analyzed using an editorial approach (e.g. [15]), meaning that the categories and statements for characterizing the reasoning in the interviews were not to a large extent predefined. The qualitative data was codified, commented and searched for patterns and themes in a first iteration. In a second step this was used to make generalizations and relate to the themes presented by Stol in his framework [17]. The mapping was then used to evaluate the compatibility of the company to adopt inner source, see Table 1. The framework was adapted with some modifications to suit the company, consisting of three additional factors located in the end of the Table 1.

2.4 Specification of requirements on technical and practical support

To define the technical infrastructure, related context and practices, a requirement specification was chosen because it is a natural approach within the software industry to describe a desired solution. The requirements specification is not presented here, although an overview of the domain can be found in Section 4, and more information is provided in [5].

Several methods were used in order to elicit requirements from all levels of interest, e.g. stakeholder analysis, additional interviews, and a series of focus groups.

Stakeholder analysis. A stakeholder analysis (e.g. [6]) was used to map all of the stakeholders and elicit their different areas of interest. It is important that

everyone with a stake in the product gets to contribute their view, goals and wishes concerning both functional and non-functional requirements in order for the final product to get a corporate wide approval.

Stakeholders were identified amongst developers, project- and service managers, team managers and corporate representatives. The analysis was based on material from the interviews held in the situation analysis, complemented by the focus group meetings and a longer interview with the case companys' former CTO.

Focus groups. As it became clear early on in the case study that stakeholders had different opinions and priorities, this technique was considered appropriate. The incentive was to create an understanding between stakeholders in addition to identify problems and gather ideas and opinions in a structured manner [6]. The other objective of the focus groups [4] was to elicit requirements for a substantial part for the proposed solution. Three areas with different themes were therefore identified on which the focus groups were based upon: Reuse of code and knowledge; Tools and functionality; Time, sales strategy and incentives. With these themes the authors regarded to have covered all relevant aspects of the product. Several subtopics were then identified around which the discussions were held.

The focus groups were carried out in 1.5 hour sessions. Focus groups 1 and 2 were both split into two sessions, while focus group 3 was carried out in one single session. The sessions were moderated by one of the two first authors, whilst the other documented by audio recording and taking supporting notes.

Each session had a brief list of subtopics where participants were allowed to briefly describe bad experiences and focus more on the ideal usage and functions. Post-it notes were used by the participants to record their opinions, where considered appropriate by the authors. These were then collected by the moderator and presented for a joint discussion. The discussions also included different aspects of risk, cost and benefits of the proposals. Where different opinions were present, a collective prioritization of the ideas was conducted and motivation to the priorities encouraged by the moderator. The sessions concluded with a summary by the moderator.

Recordings, notes and post-its were analysed by the authors using the editorial approach as was for the interviews. Iterations with clarification and elaboration was performed with the participants when needed.

3 Results from situation analysis

The situation analysis is a product of applying Stol's framework [17] to the case company. A summarized version can be found in Table 1.

As recognized before, reuse today within the division is seen as insufficient. There is no overview documentation available on what software is available for reuse, causing knowledge of what has been developed and where it is located to be spread by a "mouth-to-mouth" manner. Certain modules and functions, which

are commonly used, risk being re-developed. For example, one interviewee stated *"I use standard modules that are needed in projects that I sometimes know that someone else has done in another project or that I have done myself in another project and thereby I can use it. In other cases, we are not aware of it. Especially when a new developer enters [a project] who has not been around for so long and do not know what is available."*

It is recognized as possible to identify existing classes and functions for reuse which could constitute the initial foundation of shared assets, time is however viewed as a restricting factor. Also modularization of code is needed, which may require additional training for the developers. Another quote highlights the potential for a common framework that can be used as a standard template in many of the projects. *"We could benefit a lot from having our own demo site or basic platform, including common modules, that projects can be based on."*

An apparent need for a platform facilitating reuse and open access exist, since redundant work is conducted.

As each customer has his or her own specific requirements, potential shared assets used from project to project may have to be adapted. This allows the components to constantly evolve and mature with increased functionality, but also with the threat of increased complexity. Requirements are mainly set at the start of the project, but also constantly evolving incrementally in each sprint according to the agile scrum like process used by the teams. Releases are done frequently with each sprint. The quality assurance process with peer reviews does exist in an informal manner but needs structure and systematization.

Concerning standardized tools, one problem exists in regards with multiple version control systems with the effect of code being dispersed on multiple platforms. In general however a common set of collaborative tools are in place including an application lifecycle management tool (TeamForge[3]) which is under evaluation.

Two general and important aspects, tightly knit together, are time and budget. The time set for documentation is seldom used for its specific purpose. Transfer of knowledge in general is a subject that needs to be incorporated in the day-to-day work process in every project. There is little or no time between projects for project feedback and knowledge transfer. This would also be an issue in regards to maintenance of potential shared assets. Reason for this is to a high concern business oriented as time estimations are kept low in order to win customer deals and chargeable coverage is of high priority, leaving limited time for internal improvements.

There is an open discussions ongoing in the case company, and a willingness to change exist, even if time is a restraining factor for internal improvements as noted.

Work coordination is maintained in each project in combination with project leads and the agile process in place. Developers felt that they had an influence and that there exists an open culture. The interviews specifically highlighted the need for evangelists to overcome the responsibility issue of inner source. Potential

[3] `http://www.collab.net/products/teamforge`

evangelists could be seen within the organization, if given enough encouragement and support to take on the responsibility. This can correlate to the evangelists described in Dinkelacker et al. [1].

One last aspect identified is about code ownership. If the customer has specific ownership of the code, then the question arises whether it is okay or not for the developers to turn the code into a shared asset and reuse it in other projects. Generally it is stated in the customer contract that the case company owns the code, whilst the customer has the rights to use it. However, this is sometimes negotiated to the customer owning the code, which means the case company has no rights to reuse the solutions.

4 Overview of proposed technical infrastructure

The situation analysis showed both need and potential for the applicability of inner source. Infrastructure and defined processes are required in order to be able to create and organize shared assets, and to help facilitate inner source practices.

Based on the situation analysis, focus group meetings and additional elicitation techniques, an infrastructure comparable to that of a software forge [13] was defined and specified. The domain for the infrastructure, other than the forge, consists of the users of the forge, and the system administrator within the studied division of the case company. The systems for the developing and running customer projects, as well as the documentation of old projects, are outside the software forge domain, see Figure 2. The forge can be seen as a component library with a component project view for each shared asset, i.e. component. There are two types of components which is explained further below.

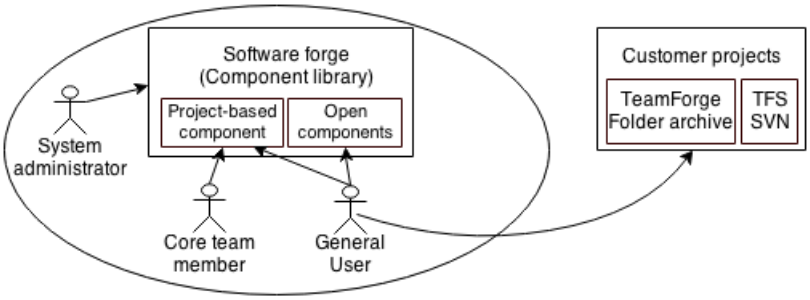

Fig. 2. Context diagram of the software forge domain and the outer domain of customer projects.

4.1 Components

All components are stored in a component library (the software forge). A search function allows the user to find a component of interest. Clear visualization

Table 1. Summary chart of findings from interviews in relation to inner source practices. Elements based on framework by Stol [17]

Element	**Findings from interviews**
Software product	
Runnable software	Classes and functions can be identified from previous projects, but the extraction may be very time consuming.
Needed by several project groups	There is potential for a common framework that can be used in several projects. An apparent need for a platform facilitating reuse exist, since redundant work is conducted.
Maturity state of the software	Constantly evolving techniques and modules for customer-specific solutions.
Utility vs simplicity	Some solutions may be too specific for the project in order to reuse.
Modularity	Modularization of code is needed, which may require training.
Development practices	
Requirement elicitation	Requirements are project-specific and are mainly set at the start of the project, but also constantly evolving in each sprint.
Implementation and quality control	Agile, sprint-driven development, planned per sprint. The level of competence in and knowledge of the process used varies. Senior developers review junior developers informally. Because of insufficient unit testing, quality can sometimes be an issue. Testing is to some extent "bazaar-like" and peer-testing is performed as much as possible.
Release management	Frequent releases, after each sprint. Customers are provided with prototypes.
Maintenance	Little or no time between projects. Development projects are generally transferred to maintenance projects after acceptance test.
Tools & Infrastructure	
Standardized tools	Common set of collaborative tools are in place, though older projects remain using older tools. Freedom to select tools locally.
Infrastructure for open access	Projects are archived in a traditional folder structure. A project platform for distributed development has been initiated and is under evaluation.
Organization & Community	
Work coordination	Developers are assigned tasks. In order to control that the correct tasks are prioritized, developers may switch between projects. Better overview desirable.
Communication	Developers sit closely together and are unlikely to benefit from "open" communication. Communication with customers is desired to be closer and steered away from e-mailing.
Leadership and decision making	Discussions are considered open and inputs appreciated. Evangelists and/or core team needed to take responsibility for a common framework.
Motivation and incentives	A lack of time is the biggest concern. Attractiveness of the tool also considered a critical success factor.
Open culture	Open discussions and willingness to change exist, though time a restraining factor for internal improvements.
Management support	Budget constraints a concern. Chargeable occupancy important. Plan for incorporating activities related to reuse in the sales strategy needed.
Additional factors	
Project feedback and knowledge sharing	More feedback on a division level desirable to improve knowledge sharing across projects.
Project initiation	Dependent on a few individuals because of expertise needed to set up projects.
Code ownership	The customer is the owner of the code, which may result in constraints on what can be reused.

of ratings and issues as well as test- and review results for each components are available so that users early can determine the potential of the component. Components can be tagged by the users with self-defined names for easier search and categorization. There are two types of components that can be shared, project-based components and open components.

Project-based components require administration by a core team that is responsible for the maintenance and development of the component. All users can access the components but the core team may restrict the user's rights to make any changes. Different types of components that have been identified as project-based components are Product, Framework and Templates.

- **Product:** A basic solution that is ready to be sold to the customer, or to be customized. This type is a long term idea and though the forge provides the infrastructure to share this component, special requirements to support this type of component will not be further considered in this project.
- **Framework:** A framework for the most commonly used modules. The framework would be used to have an initial set of modules that are reusable and can easily be implemented in a new project. A core team decides what goes into the framework and makes sure the framework is up to date and of the required quality.
- **Templates:** Documentation templates developed by a core team in order to facilitate the documentation process.

Open components do not need any administration or anyone responsible for the development of the component. No quality or generalization requirements exist to share these components and the creator is not responsible for any maintenance or further development. Modules and Classes, and Knowledge Base are types of components identified as open components.

- **Modules and classes:** Modules and classes that have been used in projects. The size and complexity of these components may vary and may be more or less suitable to reuse.
- **Knowledge base:** User guides for commonly performed tasks, tutorials, lessons learned from previous projects, common errors etc. that could be helpful in future projects.

4.2 Users

The identified user types are all employees with access to the forge. They can be divided into three groups:

- **General user:** Developers reusing and contributing to components. The general users have full rights to open components and limited rights to project-based components.
- **Core team member:** Project-based components have at least one core team member. The core team members have full access to its components and are responsible for maintenance, support and further development of that component.

– **System administrator:** The system administrators is responsible for the technical support and maintenance of the forge.

Each user will have a recorded dataset of the users activity, e.g. number of contributed tests and reviews, number of contributed components, number of components applied in customer projects.

4.3 Component project view

The components individual project views are unique and scalable, dependent of whether the component is open or project-based. Functionality as task management, issue tracking, version control, discussion areas, mailing lists and wikis are available and adapted according to each projects governance. Rating and review functionality is generically available.

5 Discussion

5.1 Infrastructure

The forge that was elicited and specified consists of two parts: The component library and component project view, which conforms to other cases e.g. SAP [13], Nokia [7, 8] and IBM [16, 19]. The former part is used for searching and finding projects, whilst the latter offers a project specific toolbox with common communication and development features where users and developers interact, also commonly identified the previously mentioned studies.

Riehle et al. [13] describes three critical design issues of a forge; finding, understanding, and contributing to projects. All three topics emerged and were heavily valued from the focus group meetings. The main feature requested for finding relevant projects was the possibility to assign self-defined tags to the projects. Concerning understandability, rating and reviews must be clearly presented to enable for a quick initial judgment. Discussion, wiki and mailing list functions described by Riehle et al [13]. was also requested to offer the user the possibility to get a more thorough understanding. Third point about the simplicity of contributing to projects regards usability and an intuitive design and integration of the different parts of the component project view.

The two types of components that were elicited, open and project-based components, offers two styles which conforms to Gurbanis [3] definitions of infrastructural and project-based inner source. Open components can be anything of general interest and the creator is not obligated to any support or maintenance of it. The main motivation for its existence is the request for simplicity to contribute and share knowledge, decrease dependency on individual developers and enable assets to be highly dynamic. Costs of creating modular and generic components need to be kept low as time is a scarce resource. For cases where more structure is needed, especially projects which are business critical and demands supervision, project-based components was introduced which are maintained and supervised by a core team. Depending on the complexity of the

component, the amount of resources needed by the core team may vary. For a critical asset such as a framework, the core team members need to have deep technical knowledge as well as an understanding of the business- and delivery models.

The mixing of these two component types offers a broad scope of possibilities for the case company. A risk however is the possibility of confusion when to choose one or the other. Another risk is that the freedom offered by open components can result in the forge being cluttered with components no one will use. This could also result in troublesome searches for the right component. One solution, which is used in the case of IBM [19], could be having an approval process where an overall supervisor decides which components gets to be added. Similar governance to that of the forge described by this paper can be identified in the case of SAP, where *"Everyone who's interested can become a developer on the forge, and everyone can register a new project without going through a lengthy approval process"* [13].

5.2 Development practices

As described in Section 4, the domain does not include the customer projects. Hence, the introduction of inner source, as proposed here, will have a low impact on the development practices used. Collaborative development is used to some extent and under improvement with the introduction of TeamForge (TF), which would benefit the implementation process of the forge.

It was revealed in the situation analysis, that the quality of the code is an issue from time to time. Hence, it is relevant to take advantage of the quality benefits associated with inner source [2,3,9,16]. With the use of ratings and peer reviews, quality can be improved both directly on the specific component and indirectly by allowing developers to gain skills through the identification of bugs and errors. A drawback of ratings, anticipated by the focus group meetings, is that it can be misleading. Users who have tested or reviewed the component may have done this to different extents why their conception of the component may vary. This is why additional information such as tagging, rating, comments and descriptions are considered important so that whole experiences are reflected and potential users can get a fair comprehension of the component. Uncertainty about the quality of a component will prohibit its use.

Alignment between the different projects is important to the team. A framework is considered to improve alignment of the development from project to project through a communal set of components, optimizing the initiation process, raising the general quality and facilitating for developers to enter a new project [1, 9].

5.3 Organization and community

Though the general organizational structure does not need to change, some effort on all levels is required to adapt to the new conditions and create business value from the initiative.

An incentive and motivational structure is essential for users to be attracted to the forge and to support the volunteer approach used, just as in any open source community [12]. From the case company managements perspective, there is a wish to acknowledge the competent developer and the platform could be used as a tool for doing so. However, it is required to put some thought into what is being measured to prevent that rewards, if any, are not misleading, nor cause negative implications for contributing. Statistics similar to what is described by Lindman et al. [8] was requested to serve as motivational data for the users directly, and also for management. This would include general measurements e.g. most popular project and more personal measures e.g. number of contributed components applied in customer projects.

The individual developer may be motivated to contribute by the rewards and acknowledgement of management, but motivation is also a highly cultural matter that needs to be incorporated in the working environment [13]. Developers as well as managers and technical leaders, need to encourage each other to contribute and to use the forge, foster awareness and integration of the solution in the day-to-day work. The need for an "evangelist" has been described in both literature [1,16] and the situation analysis, Section 3 as essential for the success of an inner source initiative. This important role is hence to be chosen carefully and early on in the initiation process in order to push the projects forward.

As described by Wesselius, [20], one of the limiting external factors of inner source development is overall profitability. That is, that the group should not optimize its own profits at the expense of the company's total profitability. By constantly keeping an awareness of what exists on the forge and the quality of it, which will be an increasing challenge as the content grows, individuals with responsibility for sales can adapt their estimates to presumptive customers and retain higher margins for tender processes [2, 3, 9, 20]. This cross-divisional dependency calls for a communication and discussion between the different internal stakeholders. Planning and development of the assets on the forge is of communal interest since it benefits the whole company.

6 Validity

The interviews were analyzed with the compatibility framework developed by Stol [17]. The framework has to our knowledge not been evaluated before by others than the author. In a later publication Stol et al. [18] presents an updated version of the framework where it has been restructured and made more generalized, still covering the same elements described in original version [17] and now applied on three new cases by the author. By using a theoretical framework the assessment is more focused, and also framed to previous findings within the research of inner source.

The proposed solution on a forge to facilitate the inner source practices and reuse was an option that emerged in the situation analysis. This platform evolved during the focus groups and continued studies of the case company, together with a clear connection to the concept of a software forge and the similarity to

Gurbani et al.'s [3] concepts of infrastructure and project-based inner source. It was not a predefined solution and evolved due to a natural process, though as no other options were as thoroughly investigated, there is a risk that alternative solutions could have been ruled out unconsciously.

Since this is an individual case study, it cannot be established if the technical solution and recommendations are applicable to other case companies. Additionally, the solution proposed has yet not been implemented, nor evaluated. The authors consider the solution to be an option for similar size of development teams, being a small company or part of a larger company that want to experiment with the adoption of inner source on a smaller scale before making significant investments. Evaluation of the solution would be needed in order to investigate what challenges the solution truly impose as well as the benefits similar organizations can expect to gain.

The main measures taken to improve the validity (e.g. [15]) in the case study can be summarized as follows.

Prolonged involvement was achieved since the two first authors spend most of their working time at the premises of the case company during a time period of about 4 months.

Peer debriefing, meaning that fellow researchers comment on the results was achieved by having the third author reviewing the findings and research methodology during the research without being actively involved in the day-to-day data collection.

Member checking, in this case meaning that senior engineers at the case company reviewed findings around which discussions were held continuously.

Audit trails were achieved by recording all interviews and taking extensive notes during data collection phases.

7 Conclusion

Several potential benefits can be gained by the case company. The main motivation is the possibilities it offers for improved reuse of code and solutions to complex problems. Other benefits include improved quality of code and general level of knowledge amongst developers, creation of a framework to standardize and shorten initiation process of new projects, better visibility and spread of information and knowledge, and higher margins for tender processes.

During this research the framework of Stol [17] was used as a framework in the analysis. We conclude that this framework is useful for this purpose, and we believe that it includes the relevant factors. There is an updated version of this framework which is more generalized [18]. We believe that by the application of the older version, this paper can also support the validity of the new version as well.

In order to address the challenges seen in introducing inner source and to gain the benefits, this case study has proposed a technical infrastructure, also known as a software forge, presented in form of a requirement specification with an adapted organizational context and work process. The forge forms a collaborative

platform where knowledge and code constitute shared assets, here known as components, and where people can interact according to the principles of inner source.

Two types of components were identified in order to address the types of data which the case company wishes to share internally. Project-based components which are, to some extent, to be seen as business critical and demands supervision by a core team. This can be related to the concepts of project-based inner source as identified by Gurbani et al. [3]. The other type, open components, relates in some parts to the concept of infrastructural inner source [3]. It can also be compared to a combination of a knowledge base and a code snippet library. This type of component can in general be anything of general interest and creator is not obligated to any support or maintenance of it.

With the specification and comprehensive description of a software forge, this paper has made a contribution as the studies available are limited and more focused on general challenges and practices. The forge presented is designed to practically handle the challenges identified and to facilitate the inner source practices of use to the studied organization. The process behind the analysis, elicitation and design of the forge can be seen as a reference for future implementations by other organizations. Focus has been from the small development team point of view, compared to global development and R&D organizations often depicted in other studies. By first understanding of how inner source can be implemented on smaller teams, the concept can be expanded with time more easily.

The division studied within the case company possesses potential for the application of inner source and if successfully applied, it can bring several rewards to the organization by optimizing its resources. An eventual future implementation is yet to be studied. Since this study has focused on one case, it cannot be generalized to other organization by default. Many of the findings though, can be of value to other cases where smaller teams and organizations are investigating the opportunities to introduce inner source, which is an area for future research.

Acknowledgments. This work was partly funded by the Industrial Excellence Center EASE - Embedded Applications Software Engineering.

References

1. Jamie Dinkelacker, Pankaj K. Garg, Rob Miller, and Dean Nelson. Progressive open source. In *ICSE '02: Proceedings of the 24th International Conference on Software Engineering*, pages 177–184, New York, NY, USA, 2002. ACM Press.
2. Vijay K. Gurbani, Anita Garvert, and James D. Herbsleb. A case study of a corporate open source development model. In *ICSE '06: Proceedings of the 28th International Conference on Software Engineering*, pages 472–481, Shanghai, China, 2006.
3. Vijay K. Gurbani, Anita Garvert, and James D. Herbsleb. Managing a corporate open source software asset. *Communication of the ACM (Association for computing machinery*, 53(2):155–159, 2010.

4. Jyrki Kontio, Laura Lehtola, and Johanna Bragge. Using the focus group method in software engineering: Obtaining practitioner and user experiences. In *International Symposium on Empirical Software Engineering*, pages 271–280, Redondo Beach, CA, USA, 2004.
5. Maria Krantz and Johan Linåker. Inner source: Application within small-sized development teams. Master's thesis, Lund University, 2012.
6. Søren Lauesen. *Software requirements: styles and techniques.* Addison-Wesley, Pearson Education Limited, Harlow, England, 2002.
7. Juho Lindman, Mikko Riepula, Matti Rossi, and Pentti Marttiin. Open source technology in intra-organisational software development: Private markets or local libraries. In Jenny S. Z. Eriksson Lundstrm, Mikael Wiberg, Stefan Hrastinski, Mats Edenius, and Pär J. Ågerfalk, editors, *Managing Open Innovation Technologies*, pages 107–121. Springer Berlin Heidelberg, 2013.
8. Juho Lindman, Matti Rossi, and Pentti Marttiin. Applying open source development practices inside a company. In *The 4th international conference on Open Source Systems*, pages 381–387, Milan, Italy, 2008.
9. Ken Martin and Bill Hoffman. An open source approach to developing software in a small organization. *IEEE Software*, 24(1):46–53, jan 2007.
10. Catharina Melian and Magnus Mahring. Lost and gained in translation: Adoption of open source software development at hewlett-packard. In *The 4th international conference on Open Source Systems*, pages 93–104, Milan, Italy, 2008.
11. Ivan Mistrík, John Grundy, André Hoek, and Jim Whitehead. Collaborative software engineering: Challenges and prospects. In Ivan Mistrík, John Grundy, André Hoek, and Jim Whitehead, editors, *Collaborative Software Engineering*, pages 389–402. Springer Berlin Heidelberg, 2010.
12. Eric S. Raymond. *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary.* O'Reilly Media, 2001.
13. Dirk Riehle, John Ellenberger, Tamir Menahem, Boris Mikhailovski, Yuri Natchetoi, Barak Naveh, and Thomas Odenwald. Open collaboration within corporations using software forges. *IEEE Software*, 26(2):52–58, 2009.
14. Colin Robson. *Real World Research.* Blackwell Publishers, 2002.
15. Per Runeson, Martin Höst, Austen Rainer, and Björn Regnell. *Case Study Research in Software Engineering: Guidelines and Examples.* John Wiley & Sons, 2012.
16. Danny Sabbah. The open internet - open source, open standards and the effects on collaborative software developmen. In *11th International workshop on High Performance Transaction Systems*, Pacific Grove, CA, USA, 2005.
17. Klaas-Jan Stol. *Supporting Product Development with Software from the Bazaar.* PhD thesis, University of Limerick, 2011.
18. Klaas-Jan Stol, Paris Avgeriou, Muhammad Ali Babar, Yan Lucas, and Brian Fitzgerald. Key factors for adopting inner source. *ACM Trans. Softw. Eng. Methodol.*, 23(2):18:1–18:35, April 2014.
19. Padmal Vitharana, Julie King, and Helena Chapman. Impact of internal open source development on reuse: Participatory reuse in action. *J. Manage. Inf. Syst.*, 27(2):277–304, October 2010.
20. Jacco H. Wesselius. The bazaar inside the cathedral: Business models for internal markets. *IEEE Software*, 25(3):60–66, 2008.