Application of inner source on small development teams - A case study

Maria Krantz, Johan Linåker

***Abstract*—The phenomenon of adopting open source software development practices in a corporate environment is also known as *inner source*. Software development is then performed in-house, imitating the way in which open source software is developed. Little has been said about the potential to adopt open source software practices in small development teams, which makes it an interesting area for research. A case study has been conducted to investigate how an organization consisting of small development teams can benefit from adopting inner source. The local division studied in the case company suffers from inadequate reuse of knowledge, which dampens productivity and implicitly results in loss of time, resources and money. The possibilities of implementing the concept of inner source to the organization were investigated and the results show that the division studied within the case company possesses potential for the application of inner source. Challenges and benefits of special importance to the case company were also identified. In order to address these challenges the case study was focused on finding both a technical and practical solution. The technical solution was developed in form of a requirement specification which describes a collaborative platform where knowledge and code can be shared, and where people can interact according to the principles of inner source. The proposed solution has yet to be implemented and evaluated in practice.**

***Index Terms*—Inner source, open source, forge, components.**

I. 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.

The phenomenon of adopting open source software development practices in a corporate environment has in research been called inner source [17], [2], community source [18], corporate open source [3], [4] and progressive open source [1]. In this report we have chosen use the term inner source, as described by Stol et al. [17]. It refers to the process of integrating open source practices within a closed, corporate environment. Software development is then performed in-house, imitating the way in which open source software is developed. Common practices of open source software development include universal access to all project artifacts, release early and often, and ”community” peer-review. The development community in inner source is within the confines of the organization, compared to open source software, where the community is external to the organization.

The research available in the area of inner source is rather limited, however, a number of case studies have been conducted of large companies that have adopted inner source. The inner source community can then be compared to that of the open source community, with a large number of developers involved. Little has been said about the potential to adopt open source software practices in small development teams, though company size in industry ranges significantly and large companies are often separated into smaller divisions with the potential of building their own communities.

A. Background

Adopting inner source requires significant effort and change management, which is 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. It is within this area that this article aims to make a theoretical contribution by conducting a case study.

An international technology consultancy 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. These features require an efficient organization where reuse of knowledge is an important corner stone.

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 consultants including developers, testers and project managers.

B. Problem description

Reuse today within the division is insufficient. Knowledge of modules and functions developed within the projects are spread in an oral and unstructured manner. The lack of a common platform, with an overview of the various modules and code bases, causes redundant work and loss of information. Hence, documentation, knowledge transfer and productivity suffers as a result.

It is within the scope of this research to evaluate whether inner source can contribute to a solution to the problem described, and if so, suggest a practical and technical solution suitable for the organization in question. The technical solution is limited to cover assets in the purpose of reuse, rather than complete customer projects.

II. Frame of reference

Due to the success of many open source projects, an interest for adopting the open source practises within companies has emerged. Mistrik et al. [9] address how closed source development 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 [19], [3], [1], [7], [11] can be found of large corporations adopting open source development.

A. Inner source models

The changes required when adopting open source practices in a corporate environment led Gurbani et al. [4] to suggest two different methods to effectively manage inner source assets; an infrastructure-based model and a project-based model. The two models are described below.

1) Infrastructure-based inner source: 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 the SourceForge[1] system. 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.

2) Project-based inner source: Gurbani et al. [4] 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.

B. Compatibility framework

Stol [17] presents a framework developed in order to assess an organization's fit for adopting inner source. The 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. The elements can be found in the left column of Table I in Appendix A.

C. Collaborative infrastructure

In open source software development, platforms, often called forges, are used to host the different open source software projects. A separation of concept is to be made between the forges hosting the projects and the platform upon which these are built. In the hosting forges, e.g. Sourceforge, developers can search amongst component projects and sign up on those he or she would like to contribute to. This part of a forge is further on described as the component library. The developer can sign up on multiple component projects and a component project view can be accessed for each project. Here the developer can take part in the projects community, follow discussions, use communal tools and download the source code.

The forge is an important concept and one of the main building parts in infrastructural inner-source. Success stories include cases from SAP [11], IBM [14], HP [1] and Nokia [7].

[1]http://www.sourceforge.com/

III. Methodology

The objective of this project is of problem solving nature. When the aim is to improve the current situation and at the same time study it, an action research methodology is appropriate [13] [20] [12]. Since this was in line with our project, this was the main method used, which consists of observing a situation or phenomena to identify the problem, find a solution to the problem and implement it, followed by an evaluation. The implementation and evaluation is yet to be performed by the case company. A combination of methods has been used for each phase according to the phase objective and the resources available.

A. Situation analysis

A situation analysis was conducted with the objective to describe and somewhat explain the current situation at the company. It can also be seen as the initial step of the requirement elicitation process. 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.

The qualitative data was collected by interviewing a sample group. The criteria for the chosen people were (i) representatives from different areas, with different work tasks, to get a good variation of answers, (ii) some work experience, within this or other companies, and (iii) availability for interviews. In total nine individuals were interviewed which included four project managers/technical leaders/senior back-end developers, two senior front-end developer, one junior front-end developer, one junior back-end developer and one service manager.

The framework presented by Stol [15], with some modifications to suit the company, were then used to evaluate the compatibility of the company to adopt inner source.

B. Requirements elicitation and specification

The result of this project is mainly a technical and a practical solution, customized after the needs of the division within the case company. A requirement specification is one way of describing the technical solution, chosen because it is a natural approach within the software industry to define a desired product. The requirements were then used to compare the products available on the market. The requirements specification is not presented in this article, though an overview of the domain can be found in Section V.

Several methods were used in order to elicit requirements from all levels of interest, e.g:

1) Stakeholder analysis: A stakeholder analysis [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.

2) 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 [5] was to elicit requirements for a substantial part of the technical solution. Three areas with different themes were therefore identified on which the focus groups were based upon; (1) reuse of code and knowledge, (2) tools and functionality, and (3) 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.

IV. Situation analysis

This situation analysis, based on the framework provided by Stol [15], has identified a number of findings showing where the case company is in the process of implementing open source practices. It also reveals in which areas improvements can be made. A summary can be found in Appendix A.

As recognized before, reuse today within the division is insufficient due to the mouth-to-mouth spread of what has been developed before and where it can be found. Certain modules and functions which are commonly used risk being redone why some work can be considered redundant. This also raises potential for a common framework that can be used as a standard template in many of the projects. Some solutions are however considered too customer specific in order to be reusable in other projects.

Two other important aspects are time and budget. These two factors are tightly knit together. The time set for documentation is seldom used for this 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. Time estimations are tight in order to win customer deals and chargeable coverage is of high priority, leaving limited time for internal improvements.

V. Overview of technical solution

The technical solution proposed is a forge platform as described in Section II-C. The domain for the technical solution consists of the forge, the users of the forge and the system administrator within the studied division of the case company. The systems for the running customer projects, as well as the documentation of old projects, are outside the domain. The forge includes a component library with a component project view for each shared component.

A. Component library

In the component library all components are stored. A search function allows the user to find a component of interest. There are two types of components that can be shared, project-based components and open components.

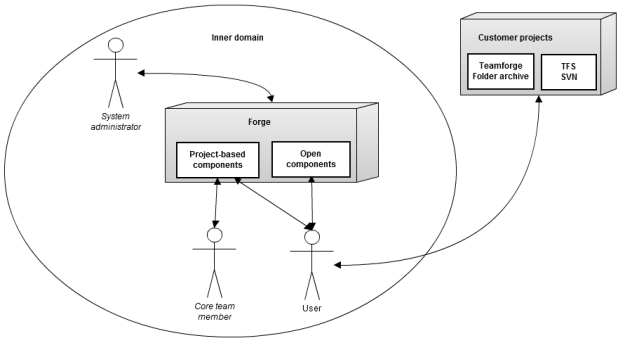

Fig. 1. Context Diagram

B. Project-based components

These 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. Types of components that have been identified as project-based components are Product, Framework and Templates.

1) 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.

2) 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.

3) Templates: Documentation templates developed by a core team in order to facilitate the documentation process.

C. Open components

The 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.

1) 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.

2) Knowledge base: User guides for commonly performed tasks, tutorials, lessons learned from previous projects, common errors etc. that could be helpful in future projects.

D. User types

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

1) General user: Developers reusing and contributing to components. The general users have full rights to open components and limited rights to project-based components.

2) 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.

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

E. Component project view

The component project view is unique and scalable, dependent of whether the component is open or project-based. Generally, according to the requirements specification, users can browse between the different versions of the component and read what changes that has been made by whom. Once the user has chosen a version of the component a snapshot can be downloaded. The user can then start investigating the component and see if it matches his expectations and requirements.

If a flaw or bug is discovered the user can report it through the issue tracker or a by submitting a comment to the component. The user can also choose to fix the bug or flaw. For this option, the user may require a complete checkout from the component repository. By using the link to the repository supplied via the user interface, the user can connect to the repository via a preferable client. When the fix is implemented and merged into the repository, the new version will be visible in the component project view for everyone else to see and download. The user is then recommended to put documentation into the component project view, e.g. via a comment or editing the wiki, so that other users can see what has been done, why and by whom.

For project-based components specifically, it is up to the core team members to decide and configure how they want the component to be managed and accessible to other users. The creator of the component is automatically assigned the role as a core team member. This individual can add additional users to the core team when appropriate. The component can be free for everyone to see and download or may be restricted to an individual or groups of users. This is to allow different levels of core team responsibility depending on the criticality of the component.

Depending on the core team's preferences and the conditions of the component, different processes can be used. A general option in open source development and closely linked to project-based inner source is that users have free access to the communal component project view where they can share information, communicate and access the repository but with reading rights exclusively. They are free to make change/feature requests and bug reports but all changes and features made by them selves are sent in as deltas in the SCM software. These deltas can then be reviewed by the core team and either be sent back or implemented into the component repository. With this way of working core team members have full control of what goes into the component and that refactorizations needed are done properly.

VI. Work process and infrastructure

The level to which inner source is adopted depends on the organization and can be done in various ways. This Section aims to provide an organizational context to the technical solution and discuss how to adopt inner source practices within the studied division, based on the situation analysis, the requirement elicitation and findings from the literature review. Specifically, the challenges and benefits of inner source found in literature have been used in order to address issues and possibilities of the work and implementation process.

A. Development practices

As described in Section V, 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. One suggestion proposed was to encourage reviews and visualizing rating and issues as well as test- and review results for each components. That way, quality can be improved both directly on the specific component and indirectly by allowing developers to gain skills through the identification of errors.

A drawback of ratings, anticipated by the team, 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.

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

B. Infrastructure

There are some possible solutions already in place which could be used to construct a forge-like environment. These alternatives, supplied by TF and TFS, are however inconsistent with the requirements that were elicited for the technical solution. Another solution is a global site available for knowledge and code sharing. This is however also in conflict with the requirement specification, especially regarding the quality requirements such as usability and maintainability.

Being one of the aims of this case study to address infrastructure, a technical solution in form of a forge was presented to the case company. The result from the screening process conducted can be of use in the work of finding a suitable solution for the final product. Alternative solutions are:

(1) TeamForge [2] which is a versatile ALM-tool already in use at the case company on a global scale. On a local level, in the division studied, it has recently been implemented for the purpose of documentation management and issue tracking within customer projects.

(2) Atlassian product pack [3] which has a diverse variety of functionality amongst its tools.

(3) Q&A on top of an ALM platform would exploit a Question & Answer system which often functions as knowledge bases where users can ask questions, answer them and rate each other's answers. A question could represent a component on a top layer which constitutes the library part of the forge. Each question links down to a project in an Application lifecycle management platform which constitutes the component project view.

(4) Custom Software is a general option available to the case company which is worth consideration in comparison to the other alternatives.

Dependent on where the division studied in the case company wants to start of, prioritization can be made differently. An introductory discussion is presented in the thesis of which this article is based on.

C. Organization and community

To complement the technical aspect of adopting inner source in the division, organizational aspects as well as community building are discussed in this section. 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.

With the solution proposed, a volunteer approach to the work coordination related to open and project-based components will be used, in contrast to assigned tasks. To encourage contributions, sharing open components does not require any further responsibility from the creator, nor a leadership or decision making structure. The motivation is to limit the responsibility of the creator, decrease dependency on individual developers and enable assets to be highly dynamic.

Regarding the project-based components, there is a need for coordination and management since these components are of a more business critical character. 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.

An incentive and motivational structure was identified in literature as essential for users to be attracted to the forge. From a management 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. Additionally, it demands of the technical solution to provide these measures on an individual developer level in a simple manner for the manager to extract.

[2]http://www.collab.net/products/teamforge

[3]http://www.atlassian.com/

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. Developers as well as managers and technical leaders, encouraging 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] [14] and the situation analysis, Section IV 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.

The uncertainty of the benefits that can be achieved makes it a difficult task to estimate how much time that is feasible to invest outside of the chargeable hours. For inner source development and the sharing of knowledge to be successful within the division, the consultants must be allowed to use time to extract code for components and to document it properly throughout the projects. Especially in the initiation, an investment of time is needed in order to get a stable system that is well tested and easy to use. As inner source demands both cultural, business and organizational changes, other than the use of a platform for sharing, it may require some time for all stakeholders involved to fully understand and adapt to the concept.

As described by Wesselius, [19], 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 retaining an awareness of what exists on the forge and the quality of it, individuals with responsibility for sales can adapt their estimates to presumptive customers. 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.

VII. Expected implications

It is found that the investigated case has good potential for adopting inner source. The division has plenty to gain by adopting the open source practises on which inner source is based, e.g.

- Improved reuse of code and solutions to complex problems [7], [1], [14], [3], [4], [19], [15] [10]
- Improved quality of code and general level of knowledge amongst developers [11], [14], [8], [3], [4]
- Creation of a framework to standardize and shorten initiation process of new projects [1], [8]
- Better visibility and spread of information and knowledge [7], [1], [14], [15], [10]
- Higher margins for tender processes [8], [3] [4], [19], [15], [10]

Emphasis should be on these benefits in the planning and evaluation processes to keep focus on what the organization want to achieve with the system.

In order to receive these benefits and prosper from them, several challenges has to be overcome. Stol et al. [17] has in

literature have identified numerous challenges, but there are some of special concern to the case company, e.g.

- Uncertainty about the quality of a component
- Awareness of the library content
- Finding the right component
- Difficulty to decide what fork or version of a components to use
- Inadequate documentation
- Motivating developers to contribute
- Commitment in terms of devoting time for the developers to contribute
- Minimum maintenance
- Costs of creating modular and generic components
- Training costs
- Uncertainty of the benefits and their value to the company

The challenges are continuous and an awareness of them throughout the organization is important in order to meet them according to the risk they impose.

VIII. Threats to validity

The literature review performed for this case study was neither systematic nor can it be established that it is complete in its coverage of the subject of inner source. The authors however have focused on peer-reviewed articles and believe to have covered all areas relevant to create the frame of reference necessary for themselves and the reader to comprehend the results presented in this article.

The situation analysis was performed in order to gain an in depth understanding of how the case company functions and an overview of its structure. As the interviews were semi-structured there was some deviation in what questions that were asked in each interview. This resulted in that some of the conclusions drawn in the situation analysis have a weaker support than others. However, considerations were made to this when identifying the problem and focus was on the main tension points that were highlighted across the board.

The interviews were analysed based on the compatibility framework developed by Stol in [15]. The framework has not been evaluated before by others than the author, but was developed based on an extensive literature study and case study within the subject performed by Stol et al. [16] [17]. For this case study it was considered a valuable tool for structuring the findings from the interviews from an inner source perspective.

The screening process, with the purpose to find an adequate prototype, focused on both COTS and OSS software. The methodology for the screening process can be considered ad hoc to some extent why it is not to be seen as neither systematic nor offering a complete coverage. Consequently, some tools of relevance could potentially have been overseen. However, the screening process constituted a foundation for the recommendations of adequate tools and was performed in relation to the requirement specification and situation analysis.

Since this is an individual case study, it can not 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.

IX. Conclusion

Several potential benefits and expected challenges for the case company have been identified. In order to address these challenges this case study has focused on finding both a technical and practical solution. A technical solution was presented to the case company in form of a requirement specification which describes a collaborative platform where knowledge and code can be shared, 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. [4]. The other type, open components, relates in some parts to the concept of infrastructural inner source. 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.

A screening process was conducted with the purpose of investigating if the the requirements could be met by products available on the market. No ready solution could be identified, but the screening resulted in four alternative solutions.

The case study has shown that the division studied possesses potential for the application of inner source and if successfully applied, it can bring several rewards to the organisation by optimizing its resources. The inner source initiative has yet to be implemented and evaluated.

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] Gary Gaughan, Brian Fitzgerald, and Maha Shaikh. An examination of the use of open source software processes as a global software development solution for commercial software engineering. In *EUROMICRO - Software Engineering and Advanced Applications (SEAA)*, pages 20–27, 2009.

[3] 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.

[4] Vijay K. Gurbani, Anita Garvert, and James D. Herbsleb. Managing a corporate open source software asset. *Community of the ACM (Association for computing machinery*, 53(2):155–159, 2010.

[5] Martin Höst, Alma Orucevic-Alagic, and Per Runeson. Usage of open source in commercial software product development - findings from a focus group meeting. In *12th International conference on Product Focused Software Development and Process Improvement*, pages 143–155, Torre Canne, Italy, 2011.

[6] Soren Lauesen. *Software requirements: styles and techniques*. Addison-Wesley, Pearson Education Limited, Harlow, England, 2002.

[7] 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.

[8] Ken Martin and Bill Hoffman. An open source approach to developing software in a small organization. *IEEE Software*, 24(1):46–53, jan 2007.

[9] 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.

[10] Lorraine Morgan, Joseph Feller, and Patrick Finnegan. Exploring inner source as a form of intraorganisational open innovation. In *The 19th European Conference on Information Systems - ICT and sustainable service development*, Helsinki, Finland, 2011.

[11] 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.

[12] Colin Robson. *Real World Research.* Blackwell Publishers, 2002.

[13] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Softw. Engg.*, 14(2):131–164, April 2009.

[14] 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.

[15] Klaas-Jan Stol. *Supporting Product Development with Software from the Bazaar*. PhD thesis, University of Limerick, 2011.

[16] Klaas-Jan Stol and Muhammad Ali Babar. Challenges in using open source software in product development: a review of the literature. In *Proceedings of the 3rd International Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development*, FLOSS '10, pages 17–22, New York, NY, USA, 2010. ACM.

[17] Klaas-Jan Stol, Muhammad Ali Babar, Paris Avgeriou, and Brian Fitzgerald. A comparative study of challenges in integrating open source software and inner source software. *Information & Software Technology*, 53(12):1319–1336, 2011.

[18] Darryl K Taft. Community source development appeals in tough times. [Mar. 15, 2012].

[19] Jacco H. Wesselius. The bazaar inside the cathedral: Business models for internal markets. *IEEE Software*, 25(3):60–66, 2008.

[20] Robert K. Yin. *Case study research: Design and methods*. Sage Publications Inc., 3rd edition, 2002.

APPENDIX A
COMPATIBILITY FRAMEWORK

TABLE I
SUMMARY CHART OF FINDINGS FROM INTERVIEWS IN RELATION TO INNER SOURCE PRACTISES.

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.
Organisation & 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.