OPAALS PROJECT

Contract n° 034824

Workpackage 12
Socio-Economic Models for Digital Ecosystems

Deliverable 12.5:

Case Studies of Inner Source

Project funded by the European Community under the "Information Society Technology" Programme

Contract Number: 034824
Project Acronym: OPAALS

Deliverable Number: Case studies of Inner Source – D12.5
Due Date: M30
Delivery Date: M33

Author: Brian Fitzgerald, Gary Gaughan, Eoin O Conchuir, Maha Shaikh University of Limerick

Partners contributed: University of Limerick

Made available to: OPAALS Community

Short Description:
The aim of this document is to provide the results of our cases studies of Inner Source to the OPAALS community.
Partners Owning: University of Limerick
Partners Contributing: University of Limerick

Versioning

Version	Date	Author, Organisation
0.1	10/7/2008	Brian Fitzgerald, Gary Gaughan, Eoin O Conchuir, Maha Shaikh University of Limerick
0.9	1/3/2009	Brian Fitzgerald, Gary Gaughan, Eoin O Conchuir, Maha Shaikh University of Limerick
1.5	17/3/2009	Brian Fitzgerald, Gary Gaughan, Eoin O Conchuir, Maha Shaikh University of Limerick

Quality check
1st Reviewer : Gabriella Lombardo, LSE
2nd Reviewer : Sotiris Moschoyiannis, LSE

Dependences:

Achievements*	Case studies of Inner source completed, with an additional, in depth study as per identified as a result of the case studies.
Work Packages	Extensive contribution to WP12 and more so to the OPAALS community as a whole, in particular pertains to the foundations of knowledge sharing and community. In addition part two specifically addresses how SMEs can address adoption of Inner Source.
Partners	CreateNet University of Kassel London School of Economics
Domains	The domains of, Inner Source, Open Source Software and Global Software Development are extensively discussed in this deliverable.
Targets	Targets of this deliverable include: Domain researchers OPAALS community Industry interest to aid future research.
Publications*	This work is at the end of the data collection and analysis phase and is yet to be published.
PhD Students*	Gary Gaughan and Eoin O Conchuir
Outstanding features*	Our understanding and level of detail of the concept of Inner Source and the companies that engage in it has fundamentally change twice in this project, we believe the work engaged in part two of this document is most mature and is of much benefit to the OPAALS community. The companies that we have worked with and not worked with (due to time constraints) and top class global software development companies and have expressed great interest in future work.
Disciplinary domains of authors*	Brian Fitzgerald, Gary Gaughan, Eoin O Conchuir, Maha Shaikh University of Limerick

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 3.0 License. To view a copy of this license, visit: http://creativecommons.org/licenses/by-nc-sa/3.0/ or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.

Table of Contents

1 Executive Summary....6

2 Introduction....7

3 Methodology....9

4 Case Studies....11

5 Inner Source Characteristics for OPAALS....18

Section II (14 in depth Research questions in a highly regulated automotive company domain)....21

Conclusion....47

References :....48

List of Figures and Tables

1 FIGURE 1 POST IMPLEMENTATION BENEFITS OF INNER SOURCE PROJECTS 11

2 FIGURE 2 MOTIVATIONS OF FIRMS FOR IMPLEMENTING INNER SOURCE.... 11

6 TABLE 1 CHARACTERIZING INNER SOURCE 17

1 Executive Summary

With our initial work on Inner Source, we examined extensively, the current activities of companies engaged in Inner source in order to arrive at the Principles of Inner Source. We feel that just as the principles of open source (Raymond, 2001) have helped in understanding, defining and implementing open source, one of the most beneficial (to the OPAALS research community, the larger research community and to companies interested in Inner Source) places to start was with the contrast between open source principles and inner source principles. However it was noted subsequently that just as can be seen in almost all domains of software engineering and also in Open Source Software, there is little evidence of firms that adhere to each and every specific of a chosen dogma or set of principles, but rather pick and choose at will, the specific principles that work best for them.

This deliverable outlines the phenomenon of Inner Source software development and places Inner Source in the context of existing open source literature. Our study includes an analysis of multiple case studies of Inner Source in use in large scale global software development companies: IBM, Hewlett Packard and Bell Labs, Nokia, Microsoft, Philips Medical and DTE Energy. The lessons learned from these case studies help us to contrast traditional open source principles with Inner Source, and we then gather the lessons to create our preliminary framework to make sense of when and how firms can adopt Inner Source. Our framework helps to make sense of the practical issues of adopting and managing Inner Source. These areas were chosen both for the background relevance to open source and emerging trends in Inner Source and also for the pertinence to the OSS research community. We have highlighted the emerging trends in the Inner Source phenomenon and surrounding areas. Awareness of this may be of great benefit to researchers in the area and industrial practitioners.

Some key results available as a result of this work include the view that social change is one of the key changes required for the application of open source methodologies within an organization. A number of other key results are available in section two, where each research question examined is summarized.

In part two of this deliverable we look in depth at one particular company, and examine 14 separate research questions as part of a study we completed in order to examine the suitability of Inner Source to one of the most regulated software domains, that of the automotive domain. We provide this deliverable with answers to each of the fourteen questions as a way to suggest beneficial changes for any organisation through applying Inner Source. The report is grounded in existing research, and we use it to suggest key actions that need to be taken when applying Inner Source in a corporate or Academic context such as OPAALS.

2 Introduction

This deliverable is in two key parts, part one represents the case studies as per the description of work, and part two involves and in depth study in a particularly challenging heavily regulated automotive domain. We examined in part one a host of companies currently employing a range of different flavours of Inner Source solutions. This enabled us to gain a great deal of general information on Inner Source solutions, however it became apparent that an in depth study was required to gain more depth to this study. The study is of course limited it that the, in depth study is of one particular company with specific operations and a specific focus.

In section one of this deliverable we outline characteristics of Inner Source extracted from our initial examination of emerging case studies and surrounding literature and subsequent interviews. Responding to the research challenges posed by Herbsleb (2007) we aim to contribute to the call to make better sense of when and how to orchestrate global development. Our detailed review of literature to date and in particular our focus on multiple case studies enable us to formulate a preliminary framework to help firms make sense of the emerging Inner Source culture, and whether it could be an appropriate choice for industry in their global development scenario. The starting point for Inner Source is an understanding of open source and its many benefits and principles. The benefits of open source (Raymond, 1999) are varying and relative to each organization. More and more we see large organizations focus heavily on open source (Dinkelacker et al 2002). We can see this focus not only by the amount of large corporate investment directed into open source projects[1], (FLOSS Report, 2006) but also by the adoption of open source methods and practices behind corporate closed doors (Gurbani et al, 2005).

Open Source is seen by many as a successful example of large scale global software development (GSD) and as such, the open source development model is attracting considerable attention as organizations seek to emulate open source success in traditional development projects, through initiatives variously labeled as ‘inner source’, ‘corporate source’ (Dinkelacker et al, 2002), ‘community source’, or ‘iSource’.

Inner source, as we use the term here, is the use of open source development methods within a corporate environment with the specific goal of attempting to benefit from the same successes achieved by open source projects. A variant of this term is progressive open source (and within it Inner Source) (Dinkelacker *et al* 2001). Inner Source is usually employed by companies to capitalize on the success that certain open source projects have enjoyed. However, there are differences between open source and closed source development and their respective development communities.

In traditional software development users and developers are typically located in separate departments and locations. This can lead to employees being unaware of development innovations and projects, and all too frequently too little mutual respect or voluntary interaction. The user-developer relationship has typically been quite different in open source. While early open source developers were users of actual products, as OSS has developed the situation has changed somewhat. In the absence of a traditional software development company, users need to become more intimately involved in the development process, as technical staff cannot simply send a checklist of requirements to the vendor to ascertain if needs, will be met. It is a widely held belief that deploying open source can lead to a sense of shared adventure which is not a common scenario in the proprietary software arena (Raymond, 2001). The transfer of these lessons, where appropriate, to conventional in-house software development is a critical activity and is of much interest to these researchers and we are eager to leverage the lessons learned here for the industrial community.

This deliverable will outline a number of case studies and exploratory interviews to give practical grounding in Inner Source. Section two of this document will examine fourteen different research questions that were examined in a specific case in SaorTech (name changed for privacy issues) an automotive company. We have worked hard to reduce the results to what is presented here. The rationale for two key parts is partially strategic and partially opportunity. Upon examining the cases studies for principles of Inner Source is became apparent that just as in Open Source there are many different flavors and strategies with Inner Source and while examining many cases enabled us to gain a breath of knowledge on the topic, the depth of knowledge could not be obtained without focusing on one domain. The domain chosen was initially to be that of the medical device industry, however due to logistical reasons a company in the automotive domain was chosen. The switch was rational as both domains are highly regulated and offer a significant challenge when introducing change.

1 “Firms have invested an estimated Euro 1.2 billion in developing FLOSS software that is made freely available. Such firms represent in total at least 565 000 jobs and Euro 263 billion in annual revenue." (FLOSS Report, 2006)

3 Methodology

Our interpretive (Walsham 1993; 1995; Lee 1994) study includes seven case studies (Yin 2002) of large global companies, Bell Labs, Hewlett Packard, IBM, DTE Energy, Nokia, Microsoft, and Philips. Case study research is appropriate when a phenomenon is complex, when a holistic, in-depth investigation is needed, and when a phenomenon cannot be studied outside the context in which it occurs (Benbasat et al. 1987, Bonoma, 1985 and Yin, 2002). Three out of the seven case study material is literature based. We reviewed both academic and industrial literature to confirm which companies were using Inner Source ideas. A thorough review of the literature and our search brought a number of companies to our attention but we wanted to focus on only those firms that had implemented Inner Source for two years or more and where there was in-depth literature available on their experience. For a number of reasons most of the literature to date has focused on somewhat successful cases of Inner Source use.

Our understanding of the literature guided our interview-based exploration into the last four cases. This deliverable is part of an ongoing study, and the process of interviewing will continue over the next six to eight months. Our initial interviews were focused around a set of semi-structured questions, and these questions emerged from our literature based study and framework.

Literature Study

Seminal work on each of the cases was identified and references followed up in academic work. Industrial reports and news articles on the same companies were scrutinized. Our data collection and data analysis stages were iterative. We built criteria for analyzing literature based on a number of categories like our definition of Inner Source, length of project duration in the company, domain area of the projects, and the size of the company (our focus is on multinational companies which tend to large in size). This list evolved over the early part of our data collection and analysis. New categories emerged over our study and we finally decided on three main cases based on our criteria and the recognition that only some cases had been academically analyzed with any detail. This provided us with a natural way to end our search and focus on three main cases.

Our analysis of the documents allowed us to immerse ourselves into the context of the company and we began to notice certain characteristics and issues for each case. There were similarities in all three cases yet each was also distinctive enough to allow our framework of principles to have wider appeal and use.

Interviews

At the time of writing this deliverable we had conducted seven key interviews, three of which were with developers and four with middle managers. The interviews were informal and were conducted with an interview guide so as to be productive in exploring the areas that were considered either to be of great interest or remained ambiguous. A typical interview lasted between thirty minutes and one hour

4 Case Studies

We examined seven different unique implementations of Inner source. The majority of implementations examined involve large-scale multinational corporations. The singular exception is DTE energy, an American energy company based in Detroit Michigan. The case studies are primarily based on literature review and supplemented by informal telephone or face-to-face interviews where ambiguity remained after the initial literature review. The preliminary interviews in some cases with public relations PR personnel were removed as no new information came to light. The remaining interviews involved developers or middle management. Developers (3) PR(8) middle managers (4)

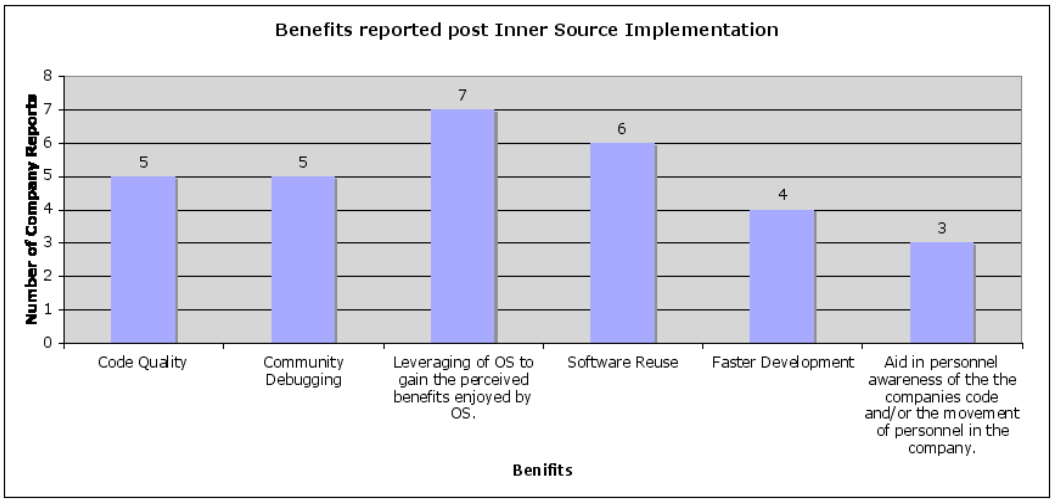

Fig. 1. Post implementation benefits of Inner Source projects.

Each of the cases was examined on how they classified (not the term Inner Source) their particular adoption of Open Source methods. Differences here obviously exist but of the cases included all fall under our umbrella term Inner Source. Other studies were examined and discounted as a result of examination. We also examined the motivational factors for attempting to implement change, the expected results versus actual, the nature of change required, problems encountered and unexpected results.

The initial list of terms was acquired on the initial examination of the case studies; this was a complete list and included every factor reported under the 5 exploratory headings (motivation, problems, benefits, project type, common theme).

These headings were chosen as many of the cases tended to focus on these particular issues. Once the categories were determined a second data collection pass was used to gather the data. A third pass confirmed the findings in case of error. The figures below were chosen as the decision was made to present only the data that was deemed by the authors roofer some insight or surprising factors. Some of the initial finding that proved insightful have been presented below in figure 1 and 2. Figure 1 illustrates the benefits reported post implementation of an Inner Source solution. In this case the highest answers being seven out of eight cases reporting the benefits to include leveraging of open source, which is hardly surprising, however the other highs include software reuse, faster development, community debugging and an increase in code quality to be benefit on Inner Source. These exercises in examining the nature of Inner Source helped guild us in the extraction of principles and characterization of Inner Source in the following sections.

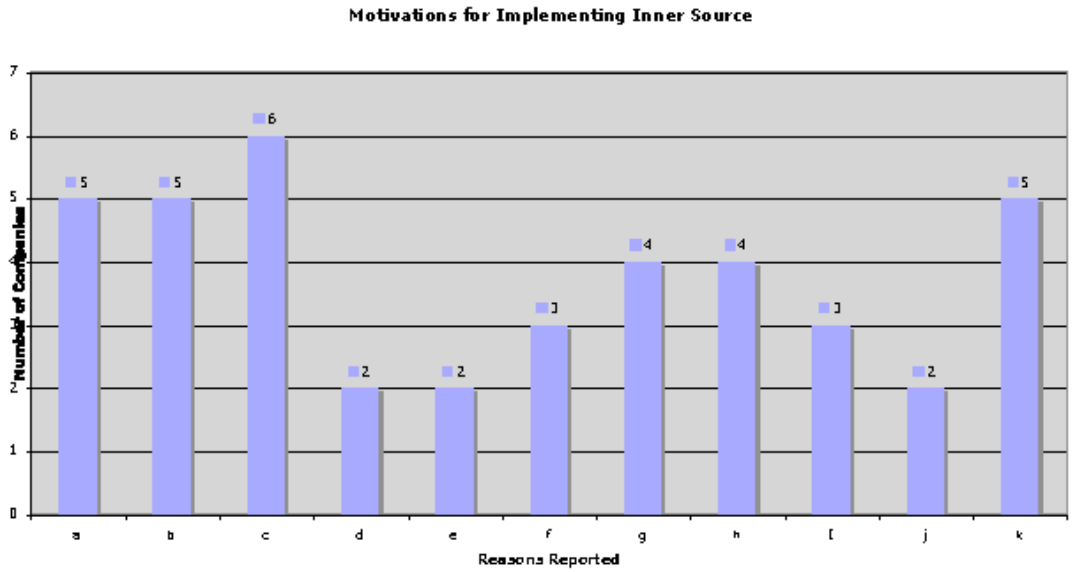

Fig. 2.1 Motivations of firms for implementing an Inner Source Solution.

The table above outlines the motivations of firm implementing Inner Source, the y-axis illustrates the number of firms reporting the individual motivating factor and the x-axis illustrates the motivating factors (A to K) involved. That is:

A	Global Software Development
B	Software Reuse
C	Overcome problems that OSS is reported to have achieved.
D	Maintenance
E	Quality
F	Many Eyeballs
G	Community Building
H	Open Discussion
I	Modular Design
J	Greater Agility
K	Decrease Costs

Fig. 2.2 Motivations of firms for implementing an Inner Source Solution. (Legend)

The most resounding motivations for getting involved in Inner Source was to improve: Global Software Development, Software Reuse, Cost Efficiency and to use OSS methods to solve company problems.

How Inner Source was applied within each company, the implications and problems that arose as a result of the implementation, the key results drawn from the case study and an analysis of each case study was the major focus of our work in this area. In certain cases the project chosen was typically an emerging area of interest in the industry with a dedicated domain expert as the champion of the project. As the project grew in maturity it would attract more attention and benefit from the snowball effect where attention would gain more developer participation and achieve a yet more mature stage.

The cases examined helped establish ongoing implementations of Inner Source and the issues surrounding such implementations. We now understand the:

1. Differences between Inner Source and traditional software development.
2. Differences between Inner Source and open source.
3. Perceived benefits of Inner Source usage.
4. Actual benefits of Inner Source usage.
5. Problems with implementing inner source solutions and some of the possible factors causing these problems.
6. Principles on Inner Source.
7. The cultural changes required for a successful Inner Source solution.

This leads us to make some arguments concerning firstly, how inner source is managed and organized and secondly, a step closer to characterizing Inner Source and formulating the main principles of Inner Source.

We examined in depth how traditional organizational and management structures have changed in order to accommodate and benefit from Inner Source adoption. The case studies indicate that there are two clear advantages of adopting Inner Source use within companies stemming from the increased visibility offered by open source principles, but also a number of problems but they are all related to the idea of visibility afforded by the open source aspect of Inner Source.

Better quality code through visibility: Visibility is an important issue to consider in Inner Source. As traditional projects move from away from traditional style development, where the code and communication is usually shared only within the project group and where face-to-face communication is common, to Inner Source style development visibility of the developer's work and textual communication increases significantly. The issues surrounding this may include an increase in quality of work as a result of many people being able to give helpful feedback but it also takes more time and effort for people to communicate in this environment.

Pride in their work through greater visibility: In HP, through increased visibility, the employees were conscious of posting to the Inner Source community as their messages were going out to a wider community. Managers felt that the use of the Inner Source solution would lead to an increase in quality due to the open nature of the Inner Source solution. People were insecure about posting their work for a company wide audience but eventually felt a lot of pride associated with contributing quality work (Melian, 2007).

Security issues surrounding visibility: However, as visibility increases so too must the companies' security efforts, especially in the case where various contractors are employed within different areas of the company. Where in the traditional development environment the access of people to code is localized within company departments and easier to manage, in contrast the very openness sought after in the use of inner source makes security and access control a lot more difficult. In HP digital badges were used to control access and vendor access to digital badges was something new that had to be managed as a result of the use of Inner Source. This creates an additional issue to be dealt with when bringing in a new vendor, as not only must the new vendor have a physical ID but also a digital badge for Inner Source access (Melian, 2007). This creates a certain level of trust within the company that the code is safe but when using external vendors the perception is that they have no control over what security the vendor is using once the vendor is given access.

Fear of job loss through visibility: Some of the fears encountered as a result in the increase in openness include fear of granting access to external access of the code to vendors and also by allowing other groups to see what we do, we run the risk of someone seeing an opportunity to downsize our department and take over our work. The other fear among developers is that, with everyone in the company potentially

having greater access to the inner workings of the company, any employee that leaves permanently, may release trade secrets to competitors and as a result threaten jobs in the company.

Privacy and knowledge retention issues related to visibility: There is also the argument that in an environment where there exists a centralized repository where a lot of information is freely available on a company wide basis and if free to traverse and browse, that this environment could make people reluctant to use digital media for some forms of communication. This was certainly the case in HP where developers rejected the suggestion of the use of video equipment to record idea generation and brainstorming in a meeting direct because of the likelihood that the media would be made freely available in a company wide system where the media could live for a significant period of time.

Increased visibility leads to easier workplace monitoring: Introduction of an Inner Source system that is perceived to be open and a controlling factor by management causes serious undermining of developers. If Inner Source is to be used effectively and to promote innovation then introduction of an Inner Source system can aid decentralized and geographically dispersed development teams to work together on a project. The openness of the Inner Source solution deployed in HP caused developers to be a lot more careful about the quality of their work and comments they were contributing. The openness of the system also caused some users to become very concerned over job security. It is clear that management and organizational structure must change and adapt to the use of Inner Source as must the users of the system.

Table 1 outlines the background difference between traditional software and Inner Source software development and the corresponding problems arising as a result of a change in environment. While some of these problems are a direct result of implementing inner source not all problems have a net negative effect. For example the problem of code forking, depending on the situation, is probably better than the previous situation where code was developed in parallel with little information about other ongoing project development. At least in the case of code forking the developers are aware of other code existing and possible solutions to existing bugs.

In this table we compare and contrast traditional software development to Inner Source software development based on the information gathered from our findings in the case studies examined, and the expectations and perceptions of the people aiming to capitalize on open source. We use this contrast of traditional software development and Inner Source software development not to suggest that the two terms are polar opposites or by implication, that traditional software development and open source software development are polar opposites, but we instead use this comparison in order to make apparent the useful areas that aid in the identification of activities that typically occur in Inner Source. There are of course cases of cross-over between both types of development for example Garcia (2007) argues that most of the characteristics found in traditional development are found in open source development, and in particular Capiluppi and Michlmayr (2007) argue that many open

source projects never migrate from centralized development to distributed development which are commonly held polar opposites between traditional development and open source development (Raymond 2001).

Another example of this cross-over is when companies attempt to build a community around a corporate product (West and O'Mahony 2004) where companies which released developed code to a public community much in the same way Nokia release code to the Maemo community.

	Traditional Development	Inner Source Development	Inner Source Problems and Issues as a result of change.
Process	Localized approach to development	Centralized approach to development.	Forking of code[1].
	Procedures and knowledge of how to account for software based on set formula assuming certain structures are present.	Absence of procedure of how to attribute value and cost when resources and input is company wide.	Inner source solutions not being valued or accounted for correctly.
	Local tools and standards based on each project preferences and closest development partners.	Company wide standards, tools and practices.	Pushing of company wide code practice, standards and tools seen as excessive and unnecessary.
	Breadth of ideas and thinking likely to be a lot smaller.	Breadth of ideas greatly increased.	Possible strain on resources as more interesting, less company centric ideas get developed.
	Defined timescale of development and testing, with complete control of the code being held by the local project group.	Constant development of code with no group having the ability to stop development for a testing cycle.	Code freezing not possible before project release.
	Communication and code access on a local level.	Open CVS, mailing lists available company wide.	Privacy of ideas and conversations an issue.
	Tight time constraints	Time constraints exist but time to market may be effected.	Possible delay in time to market.
	Much face to face communication	More towards distance interaction	Developers spend more time on communication as it is perceived that they are communicating to the entire company and the discourse can be archived.
	Heavy hierarchical business organization structures	Structured to more loose roles	Concerns over project groups taking over the responsibilities of other groups.
	Tools and methods vary from project to project.	Corporate wide policies of tools and methods can exist	Resistance to change is a real threat as tools and standards directly affect developers' day-to-day efficiency.

[1] Forking of code occurs when a copy of the code is taken and changes made without feeding back into the origon code. Usually this happens in open source when a group or sub group of developers have contrasting view of the direction of code development.

	Costing of a project is easily tracked with defined resources and costs being easily tracked with each project group.	Costing of inner source becomes much more difficult as code contributors can potentially come from any department or project	Implementing a new development model requires education of those responsible for costing as both savings and costs occur in new places and stages of development. In particular there is a cost incurred in any change applied in a company but specifically if a company wide adoption of tools and standards is required
Product	Code quality is dependent on a few key people within a project	Code quality is a concern of everyone across multiple projects, and is directly linked to the acceptance of the submitted code in the corporate repository.	The code quality required for Inner Source acceptance may be higher than is actually required and so this could have a negative effect on product development time.
	Code exists in an isolated manner with the possibility of major company resources being used to solve a problem multiple times.	Resources are less likely to be wasted on solving a problem many times as anyone can check if someone is attempting a solution with the correct communication structures in place.	There is increased dependency on technology as a medium of interaction. Allowing corporate wide access to Inner Source code means that employees who leave the company leave with more insider information than before.
	Focus on the binary shipping and release	Focus moves to producing source code for CVS submission.	Possible, delay in time to market.
	IP is a closely guarded secret	IP can be released company wide	Every employee knowing the business of the entire company a threat to jobs when that employee leaves.
	Code held in a local context	CVS and associated tool usage	Issues related to tool-use like dependence, and even political debates on which tool to adopt.
	Code review takes place at key points	Constant code review	Possible duplication of effort and inefficient use of time.

Table. 1. Framework on Inner Source Principles.

5 Inner Source Characteristics for OPAALS

- In this section we characterize Inner Source (see Table 2). Our analysis of the case studies and subsequent interviews, allowed us to organize the main arguments and issues emerging over the use of Inner Source ideas in companies. We identified these to fall within two broad categories of *product* and *process* (see Table 2). Open source can be usefully analyzed through ideas of product and process (Fitzgerald 2006) and thus this categorization is also, to a degree, useful for Inner Source studies.
- *Product* – We observed three aspects of a product that were questioned, domain area, knowledge issues and economic issues. Of these three the former two were for the most part considered unproblematic but economic issues such as the calculation of total cost of production and ownership is an area that needs further research as organizations are struggling with a way to come to an accurate estimate. Champions of Inner Source in organizations find it difficult to convince top management about the benefits of Inner Source development if they cannot justify to at least some degree the cost factor involved.
- Knowledge diffusion and innovation in the product are a true selling argument in favour of Inner Source use. The case studies indicate quite clearly that all the companies that chose to take the Inner Source route did so to enlarge he developer and knowledge base. Domain experts, through the inclusive nature of Inner Source, were able to contribute and guide project development, which lead to better quality code, and applications that were closer to user requirements.
- *Process* – Communication, knowledge, community and structure were observed to be important factors in Inner Source use. Again, we have some factors that are treated as if they are unproblematic like communication. This is how communication has been presented in the literature on Inner Source but we have reason to believe that this aspect is closely related to governance and control issues and thus not as uncontroversial as appears.
- The question of knowledge is raised again under process but the focus this time, as opposed to product knowledge where quality of code was the main concern, is on idea generation and linking this to idea dissemination and innovation. The process understanding of knowledge is much broader than the aspect looked under product. Here we show that code, knowledge and community are related and indeed inextricably linked.
- The issue of community is an important one in Inner Source and was mentioned repeatedly in the case studies examined, however, how do we define "community"? Should the "community" have certain traits in order to be able to support Inner Source? Hillery (1955) found 94 different definitions of community with only one common element, that of people. Changes in what a community is today, only serves to complicate any definition of community. The issue of community in Inner Source needs further examination and is an interesting area for future research. Questions such as how is Inner Source accepted in an organization, by

whom, what is the governance structure of the organization and how does it adapt to Inner Source, and how are smaller sub-communities of developers assimilated if they are resistant to Inner Source are all pertinent and interesting avenues for future research.

Inner Source Characteristics		
Product	*Domain area*	A common area to choose for Inner Source projects was that of research or emerging technologies.
	Economic issues	Costing of code produced in the new method was not valued correctly.
		Structural changes often would lead to reduced costs in producing a product using inner source.
	Knowledge issues	The quality of the code increased as a result of the inner source implementation.
		The use of domain experts from across the company can increase the product's quality significantly.
Process	*Communication*	CVS and associated repository tools, along with community communication tools (mailing lists, web forums, etc) were part of a common solution.
		Communication issues were common.
		Success factors that play a role in open source include simple communication mechanisms with a low learning curve.
	Knowledge issues	Computer mediated groups generate more ideas (Nunamaker *et al*, 1991).
		The breadth of ideas is a lot greater in asynchronous communications media such as is found in open source projects (Benbunan-Fich *et al*, 1999).
		Formalization inhibits innovation (Klincewicz, 2005).
		Open source development is co-evolution of code, knowledge and community.
	Community	There needs to exist a large degree of overlap between community values and governance structure.
		Motivation of company and contributors needs to be clear but crystal clear for the company.
		Open source can be described as constellations of communities who can work on what they choose rather than a formal set of tasks mandated by business objectives.
		As with implementing any change in a company and particularly open source changes to a company there is evidence to suggest that employees will resist change (Jaaksi, 2007).
	Structure	Forked code was a problem that effected support of the common infrastructure.
		Significant organizational structural changes were required.
		Structural changes often lead to reduced time to market for products using inner source.
		Boundaries are especially important, where developers can work and where contribution is not really requested.
		Open source can often be seen as an experimental approach to software development in that coordination takes place after action.

Table. 2. Characterizing Inner Source.

From our initial examination of the area and this issue in particular we have observed the community to include all developers and staff with at least code interaction duties and possibly all employees. The issue of suitable communities of Inner Source did not arise as the case studies had an inclusive approach where everyone in the company had access, and an experimental approach where they

envisaged a better product would be where everyone had input. The point was to implement the project and then evaluate the experience.

The most problematic, though interesting area under process that we found in our study is structure. This aspect is linked to all others under process but very closely to the community one. Each issue that we found related to structure has a positive and negative angle, which implies how finely tuned the governance strategy of an Inner Source policy and company will need to be. Boundaries of the project and the organization are relevant and yet often blurred because it is difficult to enclose a community that feeds (and provides food for thought) on ideas. Employees leave an organization but take a 'part' of the knowledge with them, and new people join that then need to be *included*.

As was mentioned in the HP case study, inner source methods have uses beyond software and technology specific areas. In fact elements of inner source can all readily be seen either in the OPAALS OKS or in requirements raised by the OKS user group or indeed issues raised in the OPAALS meetings. This goes beyond some of the obvious practical benefits that come with Inner source usage such as code reuse, use of common components and increase in quality of code. In particular the principles of Inner Source that stand out as being most transferable to OPAALS include: transparency and awareness of work ongoing, community, focus shift from final output submission to submitting quality work to the community at an earlier stage for constructive community input and increase in quality of work as a result. Table seven below examines in more detail some of the transferable principles of Inner Source to OPAALS.

Where Inner Source works well seems to be in areas of software, that is investigating up and coming areas of software or specifically technology research departments. Another characteristic of inner source is the interest it generates in a large group of people where a cross cutting concern or aspect is present. So in the case of Lucent everyone wanted to use the high quality product to integrate with their own product line. There were many benefits to participation and so the incentive was high. This is a very important trait, which would be of huge benefit to the OPAALS community. If similar characteristics could be identified for the research community then the authors believe that inner source could have a significant resonance within the OPAALS community.

Examples of cross cutting concerns of all researchers in OPAALS may include:

- Emerging trends in research domains
- Research methods
- Peer review
- Quality publications

Other examples of cross cutting concerns may exist in sub domains within OPAALS and these two could be capitalised on. As mentioned previously there exists a trade off between finding a common area to work together on, that is a concern or interest to many researchers, and finding a common area that is also of high value to a lot of researchers. The success factors of inner source examples however are not just limited to cross cutting concerns. There are other elements such as excellence or quality of the commons, timeliness in content and domain experts willing to improve the commons with or without immediate reward. The issue of management structure is probably the biggest issue to overcome, not in the management of OPAALS, instead the management or organisation structures inherent in the various different OPAALS researcher's domains.

Section II (14 in depth Research questions in a highly regulated automotive company domain)

It became clear upon analysis of our work in section one that and in depth case study was required to complete the classic "T" structure for our breath and depth development of knowledge of Inner Source. We initially wanted to examine the medical device industry in detail however the opportunity arose to expand into another highly regulated industry, that of the automotive domain. We worked for extended periods of time with one particular company, which shall be named SaorTech for the purposes of this report. SaorTech are very much interested in Inner Source and innovation in software development and organisational structure. At the time of the study they had a limited implementation of Inner Source in existence, however in a highly regulated and safety critical domain, where a software error could potentially result in human fatality, they are naturally cautious about extensive roll out of Inner Source practices.

We worked with the company for a number of months, to examine the current work practices in the traditional side of software development and also in the Inner Source implementation. We examined fourteen critical research questions in order to ascertain how an automotive company such as SaorTech could fully utilise Inner Source, or even if they should. While fourteen research questions initially seemed excessive, working on the exhaustive set of research questions with the company for some time failed to reduce the number of questions significantly. This is somewhat indicative of the exhaustive nature of this particular automotive working environment.

Our approach to this research is to take real examples from the OSS world to see how those projects have been successful, and how we can apply that learning in a corporate context. There are keys examples that we draw techniques from, including the Linux

and Eclipse projects. We have also looked at the most recent "inner source" literature, which has been a term adopted to label the use of Inner Source practices in the corporate context. We draw on the knowledge represented in part one of this deliverable.

Our overall approach to Inner Source in the corporate context is to create what we term a "*development community*". This community of developers should be made up of "*teams of peers*". Rather than traditional project management controls, we suggest making these development teams more independent by allowing for the development experts to make key decisions on the project. The community should be supported by communication media found in OSS projects, which include mailing lists, wikis and blogs. Your organisation should carefully study the development tools that are used by the OSS community (those tools are simple, yet powerful and highly configurable). The development community should be treated as a "*meritocracy*" where a person's success depends on the respect of their peers and their ability to work in such an open environment. A peer-based development setting will improve developer morale, and the therefore improve the efficiency of your development process. Inner Source will allow you to enforce modularity and reuse in your systems, but building on a common technical platform upon which all project can be based. Ultimately, we see that Inner Source may span across organisational boundaries to include key customers in your "*OSS ecosystem*". The development community should ultimately involve all stakeholders including your customers. This may radically change your relationship with customers, making it more inclusive and iterative.

While we have made every attempt to leverage existing research on this topic, it is a relatively new research topic with vast room for further research to allow for an increased understanding of the use of Inner Source in the corporate context. Further research should be carried out on the details of applying Inner Source, and also for wider research looking at the formation of an effective OSS ecosystem. We may also be able to learn from the active Agile development community which has already done some research on topics such as software product lines and the overlap with the Capability Maturity Model Integration (CMMI).

Product Focus

1. How good does Inner Source fit together with large number and size of requirements documents?

Outside of the corporate context, OSS projects shy away from any large requirements documents, preferring archived communications and requirement management through tools such as Bugzilla. However, in the corporate context it may be impossible to move away from large and very detailed requirements documents. With that constraint in mind, the company should adopt Inner Source within its internal processes (hidden from the customer) in order to better deal with the reality of large requirements documents. In essence, the development teams can leverage OSS-style communication channels in order to deal with the large number of requirements. All of the developers should be part of the new internal 'development community'. First, the requirements should be made available to the entire development community online. Then, discussion on them should take place on specific mailing lists, to which the community can subscribe or unsubscribe. All members of the community can refer to the mailing list archives. The discussion builds up the community's understanding of the requirements creating a shared platform of understanding. This increased shared understanding lies in with a major benefit of OSS projects, whose communities rely more on implicit understanding of the requirements rather than keeping large requirements documents up to date.

Inner Source practices deal with requirements gathering and specification better than traditional processes that require monolithic requirements documentation. Inner Source practices focus more on discussing requirements to come to a consensus, and important requirements are in fact partially implicit rather than fully explicit. Inner Source practices work because:

1. The community understands the requirements. Requirements are discussed by the community, and then accepted by consensus.
2. Requirements are not stored in large requirements documents. Rather, implicit requirements can be found in online archives of community discussions.
3. Development tools that are directly used by developers are used to capture requirements. Most significantly, the Bugzilla tool can be used not only for bug tracking, but also for tracking requirements and assigning responsibility.

This new approach to managing requirements is possible in the corporate context, but requires organisational change to allow for the 'community' of stakeholders to discuss the requirements. From the customer's side, the customer should be drawn into the 'open source ecosystem' of your company, making the requirements gathering phase more collaborative. A more iterative approach would increase the understanding of all stakeholders of the requirements being asked for. This is a major strength of OSS-type

approaches, as the entire community holds a better understanding of the implicit requirements involved. This shared knowledge can then take away part of the need for such an approach that requires huge requirements documentation.

In contrast to the world of classic software engineering, open software development communities do not seem to readily adopt or practice modern software engineering or requirements engineering processes (Scacchi 2002). However the software developed by these communities is (at least for large well-publicised projects) valuable, reliable, and trustworthy.

To answer this research question, we must first look at how the requirements engineering processes play out in OSS development projects. Scacchi (2002) examined four large open source communities with respect to their requirements engineering processes. He found that OSS projects tend to rely on implicit requirements elicitation, such as discussions found in email messages and discussion threads. Indeed, in OSS projects, there may or may not be any official requirements documents (Vidyasagar and Chang 2004). This may be because it is common in OSS to address immediate feature requirements of the user, often by the user themselves (Dinkelacker et al 2002). Ultimately, the requirements are expressed and communicated in some way on the Web. The open source approach encourages universal access to persistent information. Scacchi found that functional and non-functional requirements are formulated and agreed upon through community discussions. These discussions take place through community-accessible media including online discussion threads, community wikis, and mailing lists. These discussion platforms provide archives of discussions, and these are made freely available to the community to read. The discourse gradually builds up a shared understanding of the expected requirements, facilitated by generous hypertext cross-linking between discussions. In another example of requirements gathering, Microsoft (2007) report that potential features for their open source CodeBox product were made available to company employees online, where the best features could be voted for.

One conflict between typical OSS and traditional software engineering is that in many OSS projects, the developers are themselves users of the system being developed. As such, the requirements and design phases of OSS projects take less time to articulate and negotiate (Scacchi 2002). However, in commercial software development, the developers and users are two separate groups, and requirements must be elicited from the users (the client).

To understand the reason behind the lack of explicit documentation as described above, van Gurp (2006) suggests that the Inner Source practices are tool-centric. Developers of OSS projects are often distributed across the globe, so synchronous communication methods such as phone calls may be worthless. Instead, the development tools used are the programmers' interface to the project. As such, use

cases are rare but detailed requirements and requirements change requests are managed through the bug tracking system. van Gurp (2006) offers the following process as being typical for OSS projects, which cuts out on irrelevant feature requests: “In open source projects, slideware does not exist. New requirements for features become wiki documents. Wiki documents become bug reports. Bug reports are commented on and eventually are closed with either a reference to a patch or CVS commit or a message as to why the particular feature is no longer relevant.” For further reading, van Gurp (2006) describes three case studies of large OSS projects and how they communicate within the projects.

The question that remains for companies looking at adopting Inner Source practices is how to deal with large requirements documentation with an approach that discourages the writing of large requirements documents. Perhaps an approach for commercial projects would be to adopt the OSS mentality, and collect requirements collaboratively from the customer through shared online forms of communication such as a wiki, making requirements documents openly available for all stakeholders. These stakeholders could be allowed to comment on and discuss the shared requirement documents, helping to form a shared understanding between all stakeholders.

2. Which architectural prerequisites exist for the application of Inner Source practices?

A basic characteristic across OSS projects is the high level of modularity[2] found in the source code, and this is the primary architectural prerequisite for applying effective Inner Source practices. Effective modularity is a fundamental prerequisite for large collaborative development because it reduces coordination complexity. As such, large sections of the system can be developed in relative independence from each other. This approach will require a development platform upon which extensions can be built in modularity. A platform development group must effectively provide the framework upon which other development teams can build, and the different groups still need to coordinate between them in order to ensure interoperability. This approach also requires changes in the organisation’s control structures. Instead of a traditional top-down control structure, we suggest that the responsibility given to each development team should be maximised. Such de-centralised systems ensure great room for growth, the Internet being a prime example of such a successful system.

The Eclipse project is a major example of such work flow. The basic Eclipse platform is maintained to support all extensions that are developed for it. The platform supports a huge number of extensions and is used by many large corporations. For example, IBM’s entire toolset has been transferred from Lotus technology to be based on the

[2] A high level of modularity refers to the level of which a system such as program code can be seperated, and changes made without have an effect on the entire system.

Eclipse platform. While Eclipse is such a large project, it is experiencing enormous growth. Its modular design allows for components (which are extension or plug-ins) to be added to the project while being developed independently by large organisations. Linux is another example of how a very complex system can be developed. Its underlying system allows for different components to take on different responsibilities. For example, and Gnome and KDE projects are developed in isolation and used in many different Linux distributions, but the Linux kernel is experiencing constant growth and development.

At the heart of the 'open' approach is the concept of a community (Smith and Garber-Brown 2007). A development community can be built around the development activities of a product. In this context, we assume that the community is made up of developers. A community must be able to communicate freely in order to share ideas, in the spirit of Inner Source. The structure of the product being developed is not independent from the structure of its development community. Conway (1968) made an important realisation that the structure of the system being developed reflects the structure of the organisation that developed it. The structure of the organisation leaves a mark on the product it builds. This observation later became known as Conway's Law, and Conway stated that it was a necessary consequence of the communication needs of the people doing the work. Baldwin and Clark (2006) argue that a modular architecture is key to the success of an open source effort. MacCormack et al (2006) compared Linux code development to that of Mozilla's evolving code development. The study found that while the Linux code always showed signs of modularity, the Mozilla propriety code was much less modular. However, when the Mozilla code was later engineered, the system was made much more modular than its predecessors, and indeed more modular than the Linux architecture. It is argued that modularity is important to OSS projects since the developers involved are globally distributed and have limited means of communication. Modularity is a beneficial side-effect of this distribution.

Therefore, one must realise that the architecture of the system and the workings of the community that builds it are closely intertwined. Indeed, Dinkelacker, Garg et al. (2002) state that "*adoption of Open Source practices within a corporation is, at its heart, more of a process of social – rather than technical – change*". Smith and Garber-Brown (2007) tell their story of how DTE Energy adopted Inner Source in their corporation. A core effort in their corporation was to allow for code re-use. At first, dedicated hardware and tools were purchased to allow for easier code reuse. However, simply supplying the tools did not work. Later, the development organisation adopted a community-oriented approach, which comprised of several steps:

1. They established sessions facilitated by senior developers to allow for communication and collaboration in the community.
2. Mailing lists were established to allow participants to collaborate remotely on subtle topics. The lists were monitored, but not moderated, by senior

engineers. The mailing lists were established based on varying program perspectives, allowing the mailing list to target different slices of the community.

3. As the community matured, a wiki was established to give the developers a persistent place to share ideas an approaches. A javadoc server was also set up and was used as a central repository for all documentation.

DTE's processes benefited from the community activities. It allowed the community to form a shared language through which much debate took place on many different topics related to the development activities.

Hewlett-Packard's approach to Inner Source (termed "Progressive Open Source") in the corporate environment was published by Dinkelacker, Garg et al. (2002). With regards to the architecture, HP focused on the benefits of building a common source tree that all projects may be based upon. HP felt that Inner Source encourages the development of engineering techniques for software in an open, peer-review process. Inherent in their approach is to make all source code available to all employees behind the corporate firewall. Shared code libraries are established, and teams can work from these shared code libraries when establishing new products. In a large organisation, skilled developers can be often re-deployed to answer customer needs. A benefit of Inner Source here is that the costs of re-deployment are reduced when the developers are already accustomed to the shared source tree. In effect, Inner Source reduces the start-up time required for new teams, which is very useful in a corporate environment where quick time-to-market is critical.

As a note of caution, Vidyasagar and Chang (2004) note that in typical open source projects, maintaining software architecture is difficult because of its highly collaborative and distributive nature. Therefore, we note that in a corporate environment, the enforcement of system architecture must be stronger than in typical open source projects.

3. *How good do Inner Source practices fit together with large and sometimes monolithic legacy products?*

We can draw conclusions from IBM's experiences with their monolithic legacy systems, which were based on Lotus Notes. IBM's approach was to *evolve* their existing legacy systems into a more modern system by creating a common framework (the Eclipse project). Their legacy systems were replaced by a complete rational toolset that is now based on the Eclipse platform. As such, we can suggest that Inner Source practices need to be adopted to transform existing legacy systems into an extendable modern architecture. Of course, this is a huge undertaking for the company, and it is for the company to decide if it is worthwhile to modernise existing systems.

Unfortunately, this research question may be difficult to adequately address, as to the best of our knowledge no study has looked at Inner Source with respect to commercial legacy products. Only the possibility of refactoring legacy systems has been mentioned (such as Samoladas et al 2004). However, no research has covered how Inner Source would be integrated with existing legacy systems. Research up until now has assumed that the system being developed is new, or based on well-maintained code libraries. In brief conclusion, it has not been found that Inner Source provides any benefit when legacy products are involved; the problems with working with legacy products may not be much different when adopting Inner Source.

4. How good do Inner Source practices support software maintenance for products with long life-cycles (15 years and more)?

Inner Source practices do not offer much of a solution to help maintaining existing legacy software systems. The OSS approach is for continuous maintenance activities to take place throughout the product lifecycle. Community developers actively contribute by submitting bug reports. They may also offer software patches to the existing source code. However, apart from such activities, we do not find that Inner Source practices are a magic answer for maintaining existing legacy systems.

Applying Inner Source practices when building a project from the ground up allows for better support for maintenance. The Debian Linux project was established around 15 years ago (in 1993). We suggest adopting the approach of the Debian project, which very closely manages, and tracks the interdependencies of its components. Debian's strong process-based approach is very suitable for the corporate environment. The project maintains a constitution, a social contract, and policy documents. Debian's approach to development is to have an extensive set of policies and procedures for packing the software and all of its components. The tracking of interdependencies allow for smooth software upgrades.

When adopting Inner Source practices, we would like to draw your attention to the lifecycle of the development community itself. This community may not last 15 years. As Inner Source practices progress, we suspect that the "development community" will span organisational boundaries. As such, the communities will be comprised of multiple companies. In the book "The Living Company" by Arie de Geus, it is said that the average lifespan of a small to medium sized company is 15 years. The average lifetime of a Fortune 500 company is 40 to 50 years. The lesson from this is that we cannot assume the development community will last indefinitely. Software suppliers may drop out of the community.

In fact, it is not possible to answer this question directly because of the relative youth of the Open Source movement. The term "open source" was adopted in 1998. Before that time, there was no direct label for community development by "hackers". As such

the Open Source world has only formed in the past 10 years. We do not know how Inner Source practices support software maintenance for such a long life cycles of existing systems.

To look deeper at this question, there are two aspects to it. One aspect is the question of how to handle maintenance activities of existing legacy (long life-cycle products). The second aspect is how to design new products that are expected to have long life cycles into the future. We first need to examine how Inner Source practices address software maintenance. OSS development practice involves perpetual maintenance activities (Samoladas et al 2004). OSS development adopts an 'evolutionary model' where the software never reaches a final state, but rather keeps on evolving (Vidyasagar and Chang 2004). Whereas closed source development products often need service packs to correct existing bugs, open source development has a greater tradition of on-going bug reporting and bug fixing, which involves the entire community. This attribute of Inner Source helps to continually improve the product and to resolve bugs, and users outside of the core development teams often write bug fixes.

As a note of caution, Samoladas et al (2004) looked at the maintainability of the code produced by four large open source projects (the projects were not identified publicly). The study looked in to how the projects' code evolved over time with respect to measures related to code maintainability, such as the ratio of comments to lines of code, and the complexity of code. The study found that the maintainability of the open source projects decreased over time, just as they expected with traditional closed source projects. The authors conclude that Inner Source does not necessarily lead to easier code maintenance, and that re-factoring may eventually be needed for such projects. That being said, a separate study found that the Linux code base has been able to grow strongly at a linear rate (Godfrey and Tu 2000). This is contradictory to "Lehman's laws" that suggests that as a system grows in size, development progress slows accordingly. However, the study does not indicate what aspects of Inner Source enable strong maintenance activities of a product.

In conclusion, existing literature does not look at how Inner Source may be applied retrospectively to existing legacy systems in order to facilitate maintenance activities. However, maintenance is a core activity in OSS projects, which occurs on a continual basis. The greatest strength of Inner Source may be the availability of "many eyeballs" to continually look over others' code and to suggest improvements to be made to the system.

5. *How good do Inner Source practices support development of software with a large number of variants?*

Again, the development organisation should focus on basing its products on one common framework. Philips Healthcare (Wesselius 2008) and ICT NoviQ (Oor and

Krikhaar 2008) found that Inner Source practices strongly support platform-based development. The platform provides the base for all software variants derived from it. Extending the software platform should then develop variants of the software. The Linux community has very successfully adopted this approach. It is a huge project with many distributions (variations) and supporting software components. The common kernel supports the existence of all of the variations – it acts as the platform. One issue with developing software with variations is that bug fixes and enhancements can be applied to the variant of the software while this is not fed back to the core. Inner Source practices overcome this issue through active participation by all stakeholders. There is a tradition of passing back the improvements to allow for the core platform to be improved where possible.

There have been three academic workshops held since 2006 on the topic of Inner Source and Software Product Lines (SPL) (The International Workshop on Open Source Software and Product Lines). A related seminar, The Dagstuhl Seminar, was also held in 2008 (van der Linden et al 2008). While a traditional approach to software product families may be very different to Inner Source practices (van Gurp 2006), introducing software product-line development lets organisations better optimise software development efficiency by building a shared set of assets for reuse in multiple products (Wesselius 2008). One of the challenges with that approach is the initial creation of a shared set of reusable assets. Often, a central platform group is formed to enable the creation of the shared assets. However, it is challenging for this group to move the existing software components in use by each software group to a centralised repository. The platform group must focus attention towards the shared assets, and away from investment of existing software components. While the platform group is focused on the commoditisation of product features (a basic assumption with software product lines), each product group is more focused on getting their product to market regardless of designing for such aspects as code reuse. While Philips Healthcare (Wesselius 2008) discuss how to build a shared asset library, they do not give details on how large number of variants in products can be managed.

Chastek et al (2007) studied the Eclipse product through the lens of SPL development, and the research study offered several key findings of successful implementation of Inner Source along with SPL. The authors point to Eclipse's plug-in architecture to be the main strength of the project. The plug-in architecture allows an 'Eclipse product' to be created by using different plug-ins on top of the core Eclipse product. Another area of success is the project's high degree of modularity. The modularity makes it easier for contributors to add small improvements to the code, and makes it easier to track who is responsible for each piece of code.

van Gurp of Nokia (2006) offers key differences between Inner Source practices and SPL, and how Inner Source practices could be better adopted in an SPL environment. These lessons seem important for companies who wish to apply Inner Source in an SPL environment.

- **Tooling**. A key feature of tools in OSS projects is that the tools are developer-centric. The tools have been built over several decades for the develops' needs, such as the GCC compiler, and Bugzilla. However, in the SPL environment, there can be tools in use that are not developer-centric. For example, variability management tools can be aimed at requirements architects or even sales departments, rather than at the developers developing the software. van Gurp suggests that the lesson to be learned is that the tools in use should be able to integrate with other tools. Otherwise, information will be lost in the developers' context. Bugzilla is a good example of a tool that suits the developer. It can be integrated into version control systems and has email notification. Overall the tool supports requirements engineering, release management and even process improvement. Another issue in product family development is that as the product family grows different sets of incompatible tools can become to be relied on. In OSS, however, the tools being used are more general purpose and perhaps more basic, but are fully leveraged by the developers.
- **Code ownership**. van Gurp suggests that companies can learn from OSS projects by essentially giving more technical decision-making authority to its developers. In a 'meritocracy' found in OSS projects, the developers who are most familiar with the product make the most important decisions for the product. However, in a traditional commercial hierarchy, it may be found that incorrect decisions may be taken by people with higher authority, but who are not fully familiar with the consequences of their decisions.
- **Technical roadmap**. Similar to code ownership, van Gurp suggests that developers should be involved in the process of creating a technical roadmap. For example, refactoring may be an important aspect of developing a new version of the product, but such an activity would never be prompted by the marketing division.
- **Release management**. van Gurp suggests making a compromise with Inner Source in a commercial setting by releasing a stable version of the shared platform perhaps every two weeks. This gives a balance between open feedback by the developers using it, and for stability.

In conclusion, the intersection of Inner Source and SPL is an area that has already been studied, and is quite strong due to three research workshops being held on the area. Practitioners including Nokia have learned from existing OSS projects, and have suggested key areas that should be focused upon.

Environment Focus

6. How good do aspects of Inner Source fit with organizational factors typical to commercial environments? Relevant aspects include:

- *project management*
- *skill development*
- *career possibilities*
- *communication (e.g. in distributed development projects)*
- *motivation of project members*
- *authority, developer status, responsibility, freedom of choice*
- *fluctuation*

Project management. In order to successfully apply Inner Source practices in a corporate context, the traditional approach to project management must be aggressively modified. Instead of a traditional hierarchical top-down organisational structure, the organisation needs to be peer-centric. All peers in the development community should participate as active members of the community, and should act for their own good as well as for the good of the community at large. In OSS-style projects, decisions are more often taken by the developers who are technical experts of the system, rather than by middle management who may not be able to take the most effective decision (van Gurp 2006). Rather than traditional project management role of control, managers should instead adopt a facilitating role by facilitating the work of the developers.

Skill development. Working with an open source community gives quick and sometimes very aggressive feedback so if employees are able to cope in such a clearly meritocratic form of contribution they show skill, patience and endurance. Due to quick feedback developers do not have to wait for long before they know if they are working in a productive manner. Inner Source practices provide room for skill development. In a reputation-based ecosystem, lacking skills are made more evident due to the transparency of the process. The aggressive peer review and feedback will help developers quickly identifies areas in which they need to improve their skills (Shaikh and Cornford, 2009).

Career possibilities. At the heart of this question is employee motivation. Rather than traditional management-base possibilities, organisational change is required to support the active technical career of the developers. If we look at the top of the Maslow Pyramid of Needs, we see that job security, respect by others and problem solving are all various human needs, and this should be targeted by the career structure of the company. Inner Source practices allow developers to become more creative, and to work in a meritocracy of their peers. In the large companies that have turned to adoption of Inner Source practices we see a trend that clearly shows that in-

house employees that are creative with open source, or are able to work well with open source communities follow a similar path of career possibilities as other in-house developers as far as promotions are concerned (Shaikh and Cornford, 2009; Pulkkinen et al, 2007; Lindman et al, 2008**).** With an increased interest by companies in open source, if any developer is able to show that they have contributed to open source projects and had their ideas and code accepted, this is seen as an asset and sign of strong peer review. Such people are being hired more easily than college graduates that are not able to show a similar manner of peer reviewed work (Shaikh and Cornford, 2009).

Communication. Large and small companies use various means of communication, not all of which are based on face-to-face communication. As previously mentioned, Inner Source practices use tool-centric communication in order to maximise communication efficiency. Inner Source practices provide an overarching problem of how to coordinate the development of the system. They do this by providing open community-accessible communication media such as mailing lists and wikis, and by having a small group of core developers who choose which contributions are added to the source base, while developers can develop parts of the system in parallel thanks to the high degree of modularisation.

Motivation of project members. Open source member participation is obviously multifaceted, however a number of clear common traits exist, namely education, status and community participation. Evidence suggests that in an Inner Source implementation thus far that this holds true also and that depending on the implementation, financial reasons are also involved. In OSS projects, developers choose a project that they are most motivated by, and they contribute code for the parts of the system they are most motivated by. Therefore a corporate structure should try to address the interests of the developers, and perhaps offer flexibility in who completes which tasks.

Authority, developer status, responsibility, freedom of choice. At the heart of the OSS economy is collaboration with peers. OSS projects tend to be self-organising (rather than imposed). Authority arises from the meritocracy, and as such is based on the respect gained from one's peers. Meanwhile, as opposed to possible assumptions, freedom of choice is not a necessary trait of OSS projects. In OSS projects, decisions are taken by consensus, and are then enforced by the community. Such decisions tend not to change. Individual developers do not have limitless freedom in their coding, as their code may not be accepted by peers if it is not deemed acceptable. The ultimate freedom of choice in the OSS context is the choice to join or leave a project. In a corporate context, it is worth considering building a structure that allows developers to join and leave development teams as their interests dictate.

Fluctuation. Inner Source practices are able, through their innate flexibility to manage fluctuation in a manner that hierarchical companies cannot. Inner Source practices are

open, transparent to quite a degree, loosely coupled and based on collaboration in an agile manner that helps deal with a fluctuating external market (Butler et al, 2008). A change in the direction of code is a norm in OSSD. In fact open source thrives on different and often conflicting requirements and the process has a built in safety valve – to fork the code. Should the need arise, code can be forked and the community break apart and follow the fork or the original code. This practice is not encouraged as it can destroy critical mass of eyeballs around code but if problems become too severe then this is a healthier possibility for the code and the community (Fogel, 2005).

7. How efficiently does Inner Source practices support distributed development (and what can be adapted in conventional processes based on e.g. the V-Model)?

The OSS world is inherently distributed, and it supports distributed development very well. Since anyone can (in principle) join an OSS project, we can assume that the development community may be global (Millar et al 2005). Several studies have shown that membership of OSS projects are indeed distributed (Ågerfalk and Fitzgerald 2008). As such, OSS projects have traditionally grown to work around the issues of working in a globally-dispersed setting. Therefore we can draw lessons from Inner Source practices to be applied in the corporate world to deal with such distance. In essence, the development process needs to effectively support globally distributed communication between members of the development community. The details of such communication structures are given below.

The second half of this question asks how conventional processes such as the V-Model development may be adapted for Inner Source. We believe that the V-Model approach can be adapted for Inner Source practices irrespective of distribution. Inner Source practices' compatibility with the V-Model depends on how the V-Model has been adopted by the organisation. In OSS, there is a tendency for continuous integration, which in turn means that there is continuous verification and validation. Many OSS projects release daily builds that can be compiled each day. If the V-Model is decomposed into have the "V" activities on a daily level, then it can work very well with Inner Source practices. This continuous implementation and testing is beneficial for code quality. However, if the V-Model is applied in more of a waterfall model, we feel that this is incompatible with the Inner Source approach to software development.

The fundamental characteristic of distributed development (compared to collocated development) is that communication methods are different. In a collocated setting, colleagues can easily obtain situational awareness, letting them know if their colleague is at their desk, or perhaps look very busy. Then, face-to-face communication is possible, as you may simply walk to your colleague to ask a question. Informal non-planned communication is an important part of traditional software development – it has been found that 75 minutes of a developer's day can be

taken up by such informal contact (Perry et al 1998). However, in a distributed context, the ability for rich face-to-face contact is removed, and people must rely on information technologies to communicate such as the telephone, email, and instant messaging. Also, globally dispersed teams may cross multiple time zones, meaning the synchronous communication methods such as telephone calls are not always possible. Also, management by walking around is no longer possible (Carmel and Tija 2005).

OSS project teams have inherently had to deal with these global distances, and so methods to overcome these distances are intrinsic to the OSS approach. For communication in OSS project, developers use email and IRC as communication channels (van Gurp 2006). For technical discussions, mailing lists are used, and conversation archives are made available to the entire community. Also as previously mentioned, OSS projects are very tool-centric, and the tools used match closely to the needs of the developers. The tools are the developers' interface to the project, and so these tools must support communication-based processes. For example, Bugzilla is used to report bugs, but it supports requirements engineering moreover through feature requests being logged.

There are no studies that we found directly addressing the issue of Inner Source practices and the V-Model. However, it may be useful to look at how the agile community have adapted the V-Model, so that we can draw lessons from there. A study by Dalcher et al (2005) compared similar projects that followed different development approaches (V-model, incremental, evolutionary and Extreme Programming). The study noted the V-Model group produced a lot of requirements documentation, obviously contradicting the general approach of Inner Source practices. The study also noted that the V-Model approach involved sequential progression so that technology was only addressed during the implementations stage. Presumably, this is quite different to the approach taken with Inner Source, as the developers would be intrinsically aware of the technological details even at the outset of the project. The V-Model and similar variants are just extensions of the waterfall method (Pyhäjärvi and Rautiainen 2004). While there is some room for redefinition and redesign, it becomes more costly to make change as the process progresses. Testers become 'recommenders of change', and each change suggested may affect large parts of the system. Obviously, in line with Inner Source practices, it would be suggested to allow for testing and bug reporting throughout the entire process. Such development is quite collaborative, and patches are always welcomed from the community. Overall, this approach could greatly improve the on-going maintenance tasks of the project, ultimately giving a higher quality product.

8. How efficient do Inner Source practices support software reuse (and what can be adapted in conventional processes)?

Inner Source practices are very much about inter-organizational reuse (van Gurp 2006) and so they support software reuse very strongly. Inner Source practices support software reuse by enforcing a high degree of modularity in the system, as we have already mentioned. If we look at the Linux project, there are several degrees of reuse evident. First, the Linux kernel is used as a development platform for all of the distributions or "flavours" of Linux that are made available. By creating a solid expandable core framework, this code can be built upon to create software at higher levels of abstractions such as Fedora and Ubuntu. Second, many tools and components created for the Linux platform are made available to the community. New tools are not built from scratch, but rather use existing components as dependencies in order to add a new feature. The large Eclipse project follows the same idea of a platform and components that allows for maximise software reuse.

Software reuse is the process of creating software systems from existing software rather than building software systems from scratch. By reusing existing software code, developers can spend time on extending the components that are already available, thus potentially saving much development time. Software reuse is certainly evident in OSS projects, and has been the focus of attempts at corporate inner source.

Code modularity is a core factor of software reuse in Inner Source. Modularity means that components of the system can be designed independently but still work together to create an overall system (Baldwin and Clark 2006). A platform is also essential to provide basic architectural rules for all the modules. Modularity is essential for OSS projects, as it allows parallel development of different aspects of the system by developers who do not necessarily have regular contact with one another. Major examples of modularised OSS systems are the Linux kernel and the Eclipse project. Baldwin and Clark (2006) put forward a theory stating that high modularity in a system actually attracts more developers. The developers are happy to add to the shared code base because it gives them access to modules that others have written. They are also given an incentive to contribute code because their contribution to part of a module or as a full module will be recognised by others. Modularity makes it easier to track who is responsible for each part of the code. An underlying assumption here is that modularity enables reuse. This is true, as a module will typically be developed with a well-defined interface so that it can be integrated into the platform. By placing focus on module interfaces, the process makes it easier for these modules to be 'plugged in' to other parts of the system. Speed of development seems to require highly modularised software and small highly capable core teams and the informal style of coordination this permits (Mockus and Herbsleb 2002).

Now that we have established that modular code is a natural characteristic of OSS and that modularity is an enabler for code reuse, we must look at how such a process can

be introduced in a corporate environment. DTE Energy (Smith and Garber-Brown 2007) used Inner Source to promote software reuse in their corporation, and they claimed to have saved $600,000 USD by building a code library of more than 60 components.

Of question here is how to promote investment in shared code repositories between multiple groups in a corporate organisation. Both DTE Energy and Philips Healthcare (Wesselius 2008) provide industry experience of how this was achieved. DTE Energy's approach has already been outlined in this report, and it is of interest to look specifically at how they promoted code reuse. In their first attempt at promoting code reuse, specialised tools and server hardware were installed to enable the developers to build for code reuse. However, simply providing the tools did not succeed. Instead, DTE Energy found that building a community of developers was the solution. In essence, they found that developers must be able to communicate with one another in order to be able to share ideas. Meetings were set up between groups of developers. This allowed developers who were working on similar problems to discuss their coding. From this, code reuse was maximised because code for similar problems was only written once collaboratively, instead of many similar implementations being developed across the organisation.

At Philips Healthcare, it was also found that creating Inner Source -type development did not allow the "internal market" to automatically flourish (Wesselius 2008). They found that introducing Inner Source practices raising two issues: (1) interactions between developers and with customers were more intense, (2) making components reusable is not free, and it's not always obvious to the contributing group why they should bother making their code reusable for others. Philips Healthcare recommends introducing market mechanisms in order to create an active internal OSS-type community. They describe business models, which attempt to address this problem. A business model is comprised of four elements: product strategy, revenue logic, distribution model, and service and implementation model. An Inner Source business model should give rewards so as to encourage desired behaviour. Contributors should not feel over-burdened by the need to support the software that they contribute to the code library, so the business model needs to compensate for negative local effects. The business model also needs to give community members enough control over the software's evolution. Membership in an internal OSS-style community should bring cost- and risk-sharing, as well as indirect revenue generation based on asset contributions. Philips Healthcare's current business model involves a platform group and several systems groups. While the platform group created reusable components, it was less obvious how to get systems groups to provide reusable code. Their current business model encourages the establishment of co-development projects between developers in the platform group and developers in a systems group. The platform group bring experience in writing reusable software, while the systems group provides application knowledge ensuring that the end component addresses the requirements. The business model provides several types of "revenue" for the provider. First, it

gives the systems group more control over the deliverables that are ultimately provided by the platform group. The model also reduces the systems group's integration effort, as it ensures that the component can be matched to their specific technological needs. Finally, the platform group takes on the responsibility of maintenance and support of the component, taking the burden away from the systems group.

In conclusion, we have seen in this section of the report that Inner Source practices naturally support the modularisation of systems development, while this is a requirement for good code reuse. DTE Energy and Philips Healthcare have used Inner Source practices to promote internal software reuse. Incentives must be provided so that groups contributing code to the shared library perceive a benefit of this approach, as it can be costly to design components generically in order to allow easy reuse.

9. How efficiently do Inner Source practices (by means of e.g. online peer reviews) support quality assurance compared to conventional processes?

As the question suggests, Inner Source practices involve a strong degree of code reviewing and continuous integration to help ensure the high quality of the product released. When comparing to conventional processes, we must take into account that traditional OSS projects are not driven by deadlines but rather by a release that is of acceptable quality for the community. That being said, there are OSS projects that do stick strongly to a release timeline, notably the bi-annual release of the Ubuntu Linux distribution. Also, Inner Source practices have a higher emphasis on continuous daily releases than typical conventional processes. Continuous releases mean that there is continuous validation and verification of the product, which help to ensure the quality of the end product.

For the corporate environment to benefit from Inner Source practices, there are several characteristics that we suggest be implemented. There needs to be *strong code ownership* where there is clear responsibility held for each part of the system. The owner of the code must review any patches submitted that would modify the component. Along with continuous releases, there should be emphasis on *community-wide testing and bug reporting*. This interactive approach means that users/testers of the system will actively report bugs on the system, and will provide working patches if possible. This testing includes *code reviews* where the owners (experts) of the specific components to ensure that changes that are made do not inadvertently affect the system. While making the code open to all of the community, we do not find that there is a tradition of systemised peer reviews in the OSS community (contrary perhaps to the question that was posed). Finally, there should be a strong emphasis on *automated testing* using the correct tools that directly support the developers' work.

A key characteristic of OSS development is strong code ownership (van Gurp 2006). Many OSS projects have started with one person addressing a certain problem for their own benefit. It is this one person who generally leads the product development. While OSS projects are distributed freely and are open to modification, it is not the case that anybody can change the core product's code. It is the product owner who decides on which contributions will be accepted. They can also decide to give specific individual commit rights to the code base, but this core group of developers will generally remain small. It is this strong code ownership (enforced through a version management system) that guarantees strong code reviews and that changes are properly tested. It is in the interest of these core developers to maintain the integrity and quality of their product, and as such, they will not allow new code to be added without testing its affect on the system.

An additional aspect of quality assurance offered by OSS is that developers prioritise technical milestones, which ensure the continuing quality of their product. For example, the strong code ownership in OSS projects means that developers will plan to re-engineer older code where needed, whereas marketing requirements would never prioritise such tasks. Quality in OSS projects also tends to be high due to the release management process followed. Projects will make available alpha, beta, and release candidate versions of the product to the public. This step allows third parties to submit bugs reports and patches to the project, thus augmenting the level of quality of the project. Open Source extends the notions of inspections from a discrete, usually one or two-step process, to a continuous process, and this can potentially lead to better code quality (Dinkelacker et al 2002).

For companies wishing to apply Inner Source practices in a commercial context, the three most relevant aspects of OSS for quality assurance are large scale testing by end users; code reviews, and automated testing. The large scale testing by end users encourages early releases to the customer for testing. Bugs and missing features are entered in the bug tracking system. If the product is to be adopted by the company itself, then the source code may be made available to all developers for testing. This leads to one of the advantages of Inner Source practices: that "many eyes" test the system over the course of its development, thus highlighting all relevant bugs (Chastek et al 2007). In the spirit of OSS, developers should be free to critique others' code if this constructive criticism leads to a more efficient solution (Dinkelacker et al 2002). The company are probably already practicing automated testing, but with Inner Source practices there is a focus on the tools being used by the developers and their ability to test the code with these tools. Thus, the tools being used by the developers feed into how efficiently quality assurance processes are followed.

We can also learn from the Mozilla project, which started as a proprietary product. Due to its origins as proprietary code, the code does not reveal the same level of modularity as a traditionally OSS project such as Apache (Mockus and Herbsleb 2002). Because of the interdependence among modules in the Mozilla code,

considerable effort (inspections) must be spent in order to ensure that the interdependencies do not cause problems. Because the development teams are dependent on each other, the project has had to adopt a more formal approach to coordinating work. In Mozilla's case, this means that a single module owner must approve all changes to the module. This is a time-consuming approach, but it does allow for open contributions from the development community while restricting access to the modification of the module. The study concluded that this leads to high productivity and low defect density, but over relatively long development intervals. The fact that Mozilla was apparently able to achieve defect density levels like Apache's argues that even when an open source effort maintains much of the machinery of commercial development (including elements of planning, documenting the process and the product, explicit code ownership, inspections, and testing), there is substantial potential benefit.

10. How does the clear separation between developer and user community affect the applicability of Inner Source practices?

As the question suggests, there is a difference between typical OSS projects and more conventional design processes. OSS projects are often developed by developers wanting to address technical problems that they wish to address for their own use. Because of this, the developers develop the system to serve their needs, and so they are also the users of the system. However, in a corporate environment, developers are working on a system that they will not use. Rather, the system is being developed for a paying customer.

The main issue that the company must address when applying Inner Source practices is developer motivation. When transferring Inner Source practices from the typical OSS project to the corporate context, incentives must be provided for developers to encourage them to fully participate in the development community. A possible mechanism that we suggest is to put in place mechanisms to give developers a choice on which part of the system they wish to work on. When they do take on the work, they should be held fully responsible for that part of the system, as in OSS. Moreover, this can provide peer recognition, which can be stronger than recognition coming from management. The ultimate goal for the company should be to allow for a meritocracy of developers to form, where specific developers are recognised as being owners of different parts of the system.

While traditional software has a clear separation between developer and user it is not true that all OSS projects have this high degree of overlap of the two groups. Indeed studies indicate that the question of requirements gathering is more fluid and emergent in open source projects partly because there is a heavy overlap between users and developers (see answer to Q1 for more detail) – in effect they scratch a personal itch (Raymond, 1999). Thus if we speak of 'pure' open source communities then it is not easy to differentiate between users and developers as they so often create

software for personal use. The software grows and evolves quickly when more than one user/developer feels that this code can help solve their own problem (Ljungberg, 2000).

With an increased adoption of open source development processes, software and ideas by companies (Ågerfalk and Fitzgerald, 2008) we have noted a slight hybridization of each (Shaikh and Cornford, 2008). Criteria that are key for an Inner Source like collaboration, sharing, openness of source, testing, debugging, community effort, and 'loosely' structured governance model (O'Reilly, 1999; Capra and Wasserman, 2008; Demil and Lecocq, 2006; O'Mahony and Ferraro, 2007) are still operational when users and developers are different yet there are subtle changes (Shah, 2006).

Even when companies adopt Inner Source practices they usually do liaise with an open source community so the idea of a clear separation between both is not a reality (Shaikh and Cornford, 2009). In order to foster Inner Source practices in-house it requires a degree of inspiration and help from an open source community. The company needs to nurture the community to keep it interested in helping the company and to not view the latter as a parasite (Shaikh and Cornford, 2008). Thus if a company is serious about adopting Inner Source practices then invariably most examples reflect a need for community involvement. The community, in such cases is used as a testing ground for the software that the company is interested in developing for commercial purposes.

Most companies need to create a middle structure where their internal employees are able to communicate with developers in a manner that they are accepted by the community and not seen as outsiders (Shaikh and Cornford, 2009). This usually requires a degree of contribution by the employees towards code, advice and testing the community product (Ågerfalk and Fitzgerald, 2008). Such employees have to fit the 'profile' that makes them suitable people for such a job. Often companies choose such people from an open source community, hire them in-house, and then allow them to devote their time to creating strong links between the company and community (Shaikh and Cornford, 2009). Such developers are usually well-respected members of the community so have little trouble gaining access to the community, in spite their new role within a company.

11. Which aspects of Inner Source practices would effect the customer in a commercial environment (i.e. where would he need to adapt)?

If a company assumes an inner source approach to product development, the customer may be hidden from many of the changed processes that take place during the system development activities. There is no reason for us to believe that the customer would be negatively affected by Inner Source practices. We suggest that there may be positive effects for the customer if the OSS paradigm is applied strongly to stretch

across organisational boundaries. There are several industry examples of companies adopting an embracing approach to Inner Source practices by creating an OSS "ecosystem" which can include the company's customers. Nokia set up their open source Internet tablet platform called Maemo. This is an active open source community where the company "sources" some of its development from the open source community. Such an approach can pull customers into the OSS community by making them an integral part of the community, allowing them to interact and voice their opinions. HP (Dinkelacker et al 2002) also created an open ecosystem, an approach which they termed Progressive Open Source (POS). POS includes several layers of "openness". The inner core consists of the company's own developers share code behind a firewall. The next layer can consist of customers who have signed service agreements and non-disclosure agreements, and these customers can participate in the open community. This approach could allow the company to open its development processes to include key customers, making system development activities more interactive.

The question here is exactly who is the customer? If a company has assumed a more inner source form of approach then unless the product or service produced is open source, and importantly, distributed as well then Inner Source practices will affect the customers of this company. If there is no re-distribution this is not something that customers need to worry about. Inner source implies strongly that the product created is often NOT open source, which again brings us back to the same conclusion as above. However, in cases when some form of sourcing is adopted where an open source license is used AND the product or service is then distributed to customers then the latter will be affected in a few ways, some useful and some less so ways (Shaikh and Cornford, 2008).

Thus, under conditions when the product/service is open source and customers appropriate it, then they need to evaluate some issues such as: license; viability of community; strategic opportunities of product/service; after-sales service offered; reduced vendor lock-in; ability to innovate in-house; lack of documentation; and lack of clear contractual obligations from community in relation to support (Shaikh and Cornford, 2008; Shaikh and Cornford, 2009).

License is and always will be a slightly contentious issue depending on the degree of reciprocity required (Fitzgerald, 2006). Some companies have chosen to use a dual form of licensing to side-step concerns with GPL use (Valimaki, 2003; Ven and Mannaert, 2008) but, in general, companies favour a less reciprocal license such as the BSD or the LGPL. This ensures that they can take the open source software and mix the code with proprietary software without making the latter open source, unless they so wish (Fogel, 2005).

Viability of community refers to how sustainable is the community that is busy concentrating on the product of company interest. This is only relevant if the customer

is keen to develop the product in different and innovative ways (Hyatt and Mickos, 2008) otherwise the openness of the source implies that the customer is free to take the code and find any vendor that offers the most financially viable support. There is less vendor lock-in with open source software so customers have more options to reduce their costs without being tied into one company. After sales services are, in principle at least more flexibly sourced.

Customers also have the possibility to use open source software as a way to increase in-house development, expertise, and innovation (of software and process) (Shaikh and Cornford, 2009). Inner Source practices if appropriated in-house, usually require a degree of organizational change (Lindman et al, 2008). Such a shift can lead to greater distribution of control (Shaikh and Cornford, 2009; Pulkkinen et al, 2007), and thus a change in governance and reporting structure, a more emergent and bottom-up approach to development and innovation, greater spread of domain knowledge and expertise and more communication and collaboration within the company (Shaikh and Cornford, 2009).

Finally, a lack of or limited documentation can become an issue for customers but this is where a company offering open source products or services will ensure that this concern is addressed (Melian and Mahring, 2008; Koponen et al, 2006) by filling the gap left by an open source community (which is often reluctant to create copious amount of documentation). Customers are naturally concerned if they are required to rely on community support alone because there is no likelihood to have a contractual relationship with a community. Contracts can, and are signed with individual members of the community but as a whole a community cannot be ‘controlled’ by a company (Shaikh and Cornford, 2009).

12. Which aspects of Inner Source practices conflict with CMMI and other certification schemas?

While little research has covered CMMI+Inner Source practices, we see that these two approaches may be beneficial for each other. The CMMI community have, however, looked at how Agile development and CMMI may work together in the future in a report titled “CMMI or Agile: Why Not Embrace Both!” and we can draw some lessons from this. CMMI may be a method of bringing rigour to Inner Source approaches, which is necessary for the corporate environment. For the two approaches to work together, the Inner Source practices need to support process definition, measurement, feedback, training, and improvement in order to be compatible with CMMI. For Inner Source practices to succeed in an organisation, there must be an approach of voluntary (and incentivised) participation rather than a top-down approach that would probably not work (Wesselius 2008). We suggest that an approach combing both CMMI and Inner Source practices would need to be very

careful about allowing developers to feel that they 'participate' rather than to be forced into certain related processes.

In essence, CMMI is comprised of four areas: (1) It sets out basic rules for how to do things, (2) You must document what you do, (3) Another person reviews what you have done, (4) You evaluate the process that was followed. Upon first glance, it may seem that Inner Source practices and CMMI are not compatible with each other. For example, they are two extremes of how much documentation is involved. The Software Engineering Institute (SEI) made the same comparison between Agile development and CMMI (Software Engineering Institute 2008). The contrast comes from the two extremes, where Agile come from small fast-moving companies, with CMMI for contractually driven markets with large organisations. In this report, the SEI emphasises that CMMI is a model rather than a standard. A standard is something that is auditable, testable and compliable work on a narrow field of demonstrable outputs. However, the CMMI model contains neither processes nor procedures, but merely examples of typical work products. The process-oriented activities covered by CMMI are intended to improve real-left business activities of the organisation. The report states: "*CMMI's practices are also meant to encourage an organization to experiment with and utilize other models, frameworks, practices, and process improvement approaches, as appropriate, based on the organization's needs and priorities.*"

When using CMMI as a model, organisations are free to implement whatever applications of process improvement they want. As such, the goals of CMMI and Inner Source need not be conflicted. The report states that CMMI's objective is for the organisation to become less wasteful, leaner, and more in touch with actual development progress. When stating CMMI's objectives in such a manner, it is interesting to see that this has much overlap with Inner Source practices. Inner Source practices encourage less waste through the use of specialised configurable tools to create reusable software modules that can be reused across the organisation. CMMI is comprised of achieving goals. There are expected practices to achieve those goals, but they are not required.

SEI's report concludes "*CMMI and Agile are compatible*". We believe that this is also the case for CMMI and Inner Source. At a project level, CMMI focuses on *what* projects do, but not their development methodology. Inner Source practices focus on *how* projects are developed. SEI state that "*CMMI provides the systems engineering practices that help enable an Agile approach on large projects*", and this is probably the same for Inner Source practices. We agree with the report that CMMI can offer a "safety net" for larger projects where teams must carefully align their tasks. The report states that CMMI can offer help for larger projects that may find the following to be challenges:

- establishing overall product objectives and a project vision

- managing the allocation of requirements to teams
- defining and maintaining interfaces and constraints among teams (for both the product and the process)
- maintaining effective product integration, verification, and validation strategies for the overall product (or service)
- coordinating risk management across the project

Aspects of Inner Source

13. What is the ranking of Inner Source aspects concerning their potential to improve the efficiency of commercial software development, taking into account the constraints of commercial environments?

For this question and for Q14, we provided a list of the aspects in question, being in descending order of importance.

1. **Enforcing modularisation**, using the structure of the code to form communication paths. This allows for developers to work in parallel on different sections of the system, while reducing coordination complexity. HP report there must be a fundamental shift of seeing the ultimate output of the development process to be source code rather than binary code. By carefully designing the structure of the source code, it should be packaged as reusable components that may be used across the organisation.
2. **Tool-centric communication**, a fundamental characteristic of OSS. Because communication between distributed developers does not always suit synchronous communication methods, communication takes place through mailing lists, wikis, blogs, and through specific tools such as Bugzilla. Communication archives are kept and are available to the community for future reference.
3. **Code reuse**. In the example we have cited, DTE Energy reported $600,000 savings through code reuse. Developers should be encouraged to freely participate in relevant discussions (made possible through mailing lists). Furthermore, developers working on similar projects should be encouraged to meet with an aim to create shared reusable components.
4. **Developer motivation**. If applied correctly, developers will feel part of the development community. They will be held responsible for owning specific parts of the system, and from this peer recognition will provide feedback for their work.
5. **Common development platform**. By using simple but highly configurable tools rather than large buggy development software, the development process

including communication can take place in a tool-centric environment. Also, developers will become familiar with the shared components available. HP report that this allows for quicker ramp-up of new teams, as the developers already hold a shared understanding for the tools and components that can be used to jump-start the project.

14. What is the ranking of Inner Source aspects concerning their potential to improve the quality of software products, taking into account the constraints of commercial environments?

1. **Continuous inspections**. While not as structured as code inspections, the development community to free to review each other's source code, to report on bugs, and to contribute patches to fix bugs. Dinkelacker et al. (2002) have reported that this improves quality by ultimately shipping products with fewer defects to the customer. This also ensures higher consistency across projects (Smith and Garber-Brown 2007).
2. **Continuous integration**. Nightly builds are performed, which guarantees that problems with the integration of different components can be found almost immediately. Continuous integration implies continuous verification and validation, ensuring higher product quality.
3. **Reduced time to market**. As found by Philips Healthcare, Inner Source practices ultimately allow for a more efficient development process, thus reducing the time to market. We can argue that this is an indirect but important factor of product quality.

Conclusion

In conclusion we have examined a number of case studies involving various Inner Source solutions. We looked at the expected results of the Inner Source solution and the motivations behind the deployment. We then looked at the actual benefits and the problems associated with the deployment. The issues of managing and organizing inner source were examined. These issues are especially important to Inner Source and as we have seen some critical changes were required in the successful implementation of Inner Source. In particular we have seen that to create innovation a flexible approach to managing is required and the organizational structure must undergo some key changes to foster growth of Inner Source. One particular finding is that organizational structure is the biggest issue that must be constantly examined when implementing an Inner Source solution.

We then analyzed in detail the existing flavour of Inner Source and how they are structured. After this analysis was complete we targeted a highly regulated industry in order to examine the apparent contradiction between open source software engineering and a company adhering to a large number of software requirements and constraints. We worked in detail and in depth on a expansive range of research questions to examine how Inner Source can be used in the specific domain. While we realise that is not possible to generalise one specific case to all, it has certainly held true that one case can prove that it is in fact (at least in certain circumstances) possible to implement Inner Source in this specific environment.

This of course has specific implications for OPAALS. The research has shown that many different flavours exist of Inner Source, and the key to success is choosing the ones specifically that make sense and work for each environment. Likewise, a number of experiences are available from our research, that suggest what may or may not work as a solution, and the most likely outcomes or traits that exist for specific styles of Inner Source. This of course has already been seen in the OPAALS community with elements of knowledge sharing and communication. It is as always up to the community to decide what changes if any would be welcomed and we offer our work here in Inner Source as a guide to further develop the OPAALS community.

References :

Ågerfalk, P. J. & B. Fitzgerald (2008) 'Outsourcing to an unknown workforce: Exploring open sourcing as a global sourcing strategy', *MIS Quarterly*, 32 (2)**,** 385-409.

Ågerfalk P J (2004). Investigating Actability Dimensions: A Language/Action Perspective on Criteria for Information Systems Evaluation, Interacting with Computers, 16(5), pp. 957–988.

Ågerfalk, P.J. (2005). Studying Persistent Conversations in an Open Source Context: A Conceptual Framework, In Proceedings of Promote IT 2005, (Eds, Bubenko jr., J et al.) Studentlitteratur, Lund.

Arlein, R. and Gurbani, V., An Extensible Framework for Constructing Session Initiation Protocol (SIP) User Agents. Bell Labs Technical Journal, 9, 3 (November 2004), p. 87-100.

Benbasat, I, Goldstein, D.K., and Mead, M. (1987) "The Case Research Strategy in Studies of Information Systems", MIS Quarterly, 11:3, pp. 369-385.

Bonoma, T.V. (1985) "Case Research in Marketing: Opportunities, Problems and a Process" Journal of Marketing Research (22), pp. 199-208

Benbunan-Fich, R., and Roxanne Hiltz, S. "Impacts of Asynchronous Learning Networks on Individual and Group Problem Solving: A Field Experiment," Group Decision and Negotiation (8) 1999, pp 409-426.

Bergquist, M. and Ljungberg, J. "The Power of Gifts: Organising Social Relationships in Open Source Communities" Information Systems Journal (11:4) December, 2004 pp.305-320.

Bezroukov, N. (1999a) "Open Source Software Development as a Special Type of Academic Research (Critique of Vulgar Raymondism)", *FirstMonday: Peer Reviewed Journal on the Internet,* **4 (10),** pp. 1-23.

Bezroukov, N. (1999b) "A Second Look at the Cathedral and the Bazaar", *FirstMonday: Peer Reviewed Journal on the Internet,* **4 (12),** pp. 1-29.

Capek, P.G., Frank, S.P., Gerdt, S. and Shields, D. (2005) A history of IBM's open-source involvement and strategy, IBM Systems Journal, 44, 2, 249-257.

Capiluppi, A. and M. Michlmayr (2007) 'From the Cathedral to the Bazaar: An Empirical Study of the Lifecycle of Volunteer Community Projects' in IFIP International Federation for Information Processing, Volume 234, Open Source Development. Adoption and Innovation, eds. J. Feller, Fitzgerald. B., Scacchi, W., Sillitti, A. Boston: Springer, pp. 31-44.

Carmel, E. and Agarwal, R. (2001). Tactical approaches for alleviating distance in global software development. IEEE Software, 18 (2), 22-29.

Carroll, J. M. (1996). "Becoming Social: expanding scenario-based approaches in HCI." Behaviour & Information Technology 15(4): 266–275.

Clark, H. H. (1996). Using Language. Cambridge, Cambridge University Press.

Conway, M.E., How Do Committees Invent? Datamation, 14, 4 (1968), p. 28-31.

Crowston, Kevin, Scozzi, Barbara, Open source software projects as virtual organizations: Competency rallying for software development. IEE Proceedings Software, 149(1), 3-17. (2002).

Cubranic, D. (1999). Open-Source Software Development. In 2nd Workshop on Software Engineering over the Internet, 7p.

Curium, F. and J. A. Holstein (Eds.) (2001) *Handbook of Interview Research: Context and Method,* Sage Publications.

Cusumano, M. and R. Selby (1997) "How Microsoft Builds Software", Communications of the ACM, 40 (6), pp. 53-61.
Gallivan, M.J. "Striking a Balance between Trust and Control in a Virtual Organization: A Content Analysis of Open Source Software Case Studies," Information Systems Journal (11:4) 2001, pp 277-304.

Gurbani, V., Garvert, A., and Herbsleb, J., A Case Study of Open Source Tools and Practices in a Commercial Setting, Proceedings of the 5th ACM Workshop on Open Source Software Engineering (WOSSE), (May 2005), pp. 24-29.

Dinh-Trong, T. and Bieman, J.M. (2004) Open Source Software Development: A Case Study of FreeBSD. In Proceedings of the 10th International Symposium on Software Metrics, IEEE Computer Society.

Dinkelacker, J., Garg, Corporate Source: Applying Open Source Concepts to a Corporate Environment. HP Technical Report 2001. http://www.hpl.hp.com/techreports/2001/HPL-2001135.html?jumpid=reg_R1002_USEN

Dinkelacker, J., Garg, P., Miller, R., and Nelson, D., Progressive Open Source. HP Technical Report 2001. http://www.hpl.hp.com/techreports/2001/HPL-2001-233.html.

Dinkelacker, J., Garg, P., Miller, R., and Nelson, D., Progressive Open Source. In Proceedings of the 2002 ACM International Conference on Software Engineering (ICSE'02), pp. 177-184, May 2002.

Erenkrantz, J.R. and Taylor, R.N. (2003). Supporting Distributed and Decentralized Projects: Drawing Lessons from the Open Source Community. In The 1st Workshop on Open Source in an Industrial Context, p. 21-30, <wwwbib.informatik.tu/muenchen.de/infberichte/2003/TUM-I0319.pdf>

Fielding, R.T and Kaiser, G. (1997). The Apache HTTP server project. IEEE Internet Computing, 1 (4), 88-90.

FLOSS Report EU http://ec.europa.eu/enterprise/ict/policy/doc/2006-11-20-flossimpact.pdf

Gacek, C. and Arief, B. (2004). The Many Meanings of Open Source. IEEE Software, 21 (1), 34-40.

Gallivan, M.J. "Striking a Balance between Trust and Control in a Virtual Organisation: A content Analysis of Open Source Software Case Studies" Information Systems Journal (11:4) December, 2004 pp.277-304.

Garcia, M.J. and E.W Steinmueller (2003) 'The Open Source Way of Working: a New Paradigm for the division of labour in Software development?' INK Open source working paper No.92, SPRU Science and Technology Policy Research, University of Sussex, available at: http://www.sussex.ac.uk/spru/1-6-1-2-1.html, last accessed 15/08/07.

German, D.M. (2003). GNOME, a case of open source global software development. In International Workshop on Global Software Development, p. 39-43, <gsd2003.cs.uvic.ca/gsd2003proceedings.pdf>

Goffman E (1959) "The Presentation of Self in Everyday Life," Doubleday: Garden City, New York.

Goldkuhl, G. and P. J. Ågerfalk (2002). Actability: a way to understand information systems pragmatics. In Coordination and Communication Using Signs: Studies in Organisational Semiotics 2.

K. Liu, R. J. Clarke, P. B. Andersen and R. K. Stamper. Boston, Kluwer Academic Publishers: 85–113.

Grand, S., G. von Krogh, D. Leonard, and W. Swap (2004). Resource allocation beyond firm boundaries: A multilevel model for open source innovation. Long Range Planning 37, 591–610.

Gurbani, V.K., Garvert, A. and Herbsleb, J.D. (2005). A Case Study of Open Source Tools and Practices in a Commercial Setting. In Proceedings of the 5th Workshop on Open Source Software Engineering, p. 24-29, ACM.

Halloran, T.J. and Scherlis, W.L. High Quality and Open Source Practices. in Meeting Challenges and Surviving Success: 2nd Workshop on Open Source Software Engineering. 2002. Orlando, FL.

Herbsleb, J. (2007) "Global Software Engineering: The Future of Socio-Technical Coordination". in *Future of Software Engineering FOSE '07,* pp. 188-198, IEEE Computer Society.

Herbsleb, J.D. and Grinter, R.E., Architectures, Coordination, and Distance: Conway's Law and Beyond. IEEE Software, Sept./Oct., (1999), p. 63-70.

Herbsleb, J.D. and Mockus, A., An Empirical Study of Speed and Communication in Globally-Distributed Software Development. IEEE Transactions on Software Engineering, 29, 3 (2003), p. 1-14.
Hillery, G. A. (1955) "Definitions of Community: Areas of Agreement", *Rural Sociology,* **20** pp. 111-123.

IBM Community Source Interview, http://www.betanews.com/article/IBM_Turns_to_Open_Source_Development/1118688437 2007.

Jaaksi, Ari (2007) 'Experiences on Product Development on Open Source Software' in IFIP International Federation for Information Processing, Volume 234, Open Source Development. Adoption and Innovation, eds. J. Feller, Fitzgerald. B., Scacchi, W., Sillitti, A. Boston: Springer, pp. 85-96.

Kirsch, L.J "Deploying Common Systems Globally: The Dynamics of Control" Information Systems Research (15:4), December 2004, pp. 375-395.

Klincewicz K. (2005), Innovativeness of Open Source software projects. MIT Working Papers.

Krishnamurthy, S. (2002). Cave or Community? An Empirical Examination of 100 Mature Open Source Projects. First Monday, 7 (6), <www.firstmonday.org/issues/issue7_6/krishnamurthy/>

Lee, A.S. "Electronic Mail as a Medium for Rich Communication: An Empirical Investigation Using Hermeneutic Interpretation," MIS Quarterly (18:2), June 1994, pp. 143-157.

MacCormack, A., Rusnak, J., and Baldwin, C., Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code, in Harvard Business School Working Paper. 2004: Boston, MA 02163.

Melian, C. Progressive Open Source, PhD Thesis, Stockholm School of Economics 2007.

Michlmayr, M. Quality Improvement in Volunteer Free and Open Source Software Projects, PhD Thesis, University of Cambridge 2007.

Mockus, A. and Herbsleb, J.D. (2002). Why Not Improve Coordination in Distributed Software Development by Stealing Good Ideas from Open Source?. In Meeting Challenges and Surviving Success: The 2nd Workshop on Open Source Software Engineering, p. 19-25, <opensource.ucc.ie/icse2002/MockusGerbsleb.pdf>

Mockus, A., Fielding, R., and Herbsleb, J.D., Two Case Studies of Open Source Software Development: Apache and Mozilla. ACM Transactions on Software Engineering and Methodology, 11, 3 (2002), p. 309-346.

Moon, J.Y. and Sproull, L. (2000). Essence of Distributed Work: The Case of the Linux Kernel. First Monday, 5 (11), <www.firstmonday.org/issues/issue5_11/moon/>

Nunamaker, J. F. Dennis, Alan R. Valacich, Joseph S. Vogel, Douglas. George, Joey F. Electronic meeting systems, Communications of the ACM, v.34 n.7, p.40-61, July 1991.

Open Source Business, The Economist, March 16th 2006 http://www.economist.com/business/displayStory.cfm?story_id=5624944

Raymond, E. (1999) *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary,* O'Reilly & Associates, Sebastopol, California.

Rosenberg, J., et al., SIP: Session Initiation Protocol, IETF RFC 3261, July 2002, http://www.ietf.org/rfc/rfc3261.txt

Sagers, G "The Influence of Network Governance Factors on Success in Open Source Software

Development Projects. Proceedings of the Twenty-Fifth International Conference on Information Systems.

Agarwal, R and Kirsch, L. (eds.) Washington DC, December 2004, pp.427-438.

Sarma, A. (2005). A Survey of Collaborative Tools in Software Development, ISR Technical Report # UCI-ISR-05-3, Institute for Software Research, University of California, Irvine <www.isr.uci-edu/tech-report.html>

Scacchi, W., Understanding the requirements for developing open source software systems. IEE Proceedings on Software,149, 1 (February 2002), p. 24-39.

Shah, S. Understanding the Nature of Participation and Coordination in Open and Gated Source Software Development Communities. In Annual Meeting of the Academy of Management. 2004.

Walsham, G. *Interpreting Information Systems in Organizations*, Wiley, Chichester, 1993.

Walsham, G. "Interpretive case studies in IS research: nature and method," *European Journal of Information Systems* (4), 1995, pp. 74-81.

West, J. and C.S. O'Mahony (2004). 'Contrasting Community Building in Sponsored and Community Founded Open Source Projects', available at: http://opensource.mit.edu/papers/westomahony.pdf , last accessed 06/08/07.

Woods, D., and Guliani, G. Open Source for the Enterprise, O'Reilly Media, Sebastopol, CA, 2005.

Yin, R. K. Case Study Research, Design and Methods, 3rd ed. Newbury Park, Sage Publications, 2002.

Baldwin, C. Y. & K. B. Clark (2006) 'The Architecture of Participation: Does Code Architecture Mitigate Free Riding in the Open Source Development Model?' *Management Science*, 52 (7)**,** 1116-1127.

Carmel, E. & P. Tija (2005) *Offshoring Information Technology : Sourcing and Outsourcing to a Global Workforce*, Cambridge, United Kingdom: Cambridge University Press.

Chastek, G. J., J. D. Mcgregor & L. M. Northrop (2007).'Observations from Viewing Eclipse as a Product Line', in *Second International Workshop on Open Source Software and Product Lines*, Limerick, Ireland, June 14,

Dalcher, D., O. Benediktsson & H. Thorbergsson, 2005. Development life cycle management: a multiproject experiment. *Engineering of Computer-Based Systems, 2005. ECBS '05. 12th IEEE International Conference and Workshops on the.*

Godfrey, M. W. & Q. Tu, 2000. Evolution in open source software: a case study. *Software Maintenance, 2000. Proceedings. International Conference on.*

Microsoft Corporation, 2007. Open Source at Microsoft (CodeBox: Bringing the Open Source Approach In-House).

Millar, C., C. J. Choi, E. T. Russell & J.-B. Kim (2005) 'Open Source Communities: An Integrally Informed Approach', *Journal of Organizational Change Management*, 18 (3)**,** 259 - 268.

Mockus, A. & J. D. Herbsleb (2002) 'Two case studies of open source software development: Apache and Mozilla', *ACM Transactions on Software Engineering Methodology*, 11 (3)**,** 309-346.

Oor, P. & R. Krikhaar (2008).'Balancing Technology, Organization, and Process in Inner Source', in BERMEJO, J., LUNDELL, B. & VAN DER LINDEN, F., eds., *Combining the Advantages of Product Lines and Open Source*, Dagstuhl, Germany, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany,

Perry, D. E., G. S. Gil & L. G. Votta (1998).'A Case Study of Successful Geographically Separated Teamwork', in *Software Process Improvement 1998 (SPI98)*, Monte Carlo, December 1998,

Pyhäjärvi, M. & K. Rautiainen (2004) 'Integrating Testing and Implementation into Development', *Engineering Management Journal*, 16 (1)**,** 33-39.

Samoladas, I., I. Stamelos, L. Angelis & A. Oikonomou (2004) 'Open source software development should strive for even greater code maintainability', *Communication of the ACM*, 47 (10)**,** 83-87.

Scacchi, W. (2002) 'Understanding the requirements for developing open source software systems', *IEEE Software*, 149 (1)**,** 24–39.

Smith, P. & C. Garber-Brown, 2007. Traveling the Open Road: Using Open Source Practices to Transform Our Organization. *AGILE 2007.* Washington D.C.

Software Engineering Institute, 2008. CMMI or Agile: Why Not Embrace Both! , Software Engineering Institute.

Van Der Linden, F., J. Bermejo & B. Lundell, 2008. Abstracts Collection. IN VAN DER LINDEN, F., BERMEJO, J. & LUNDELL, B. (Eds.) *Combining the Advantages of Product Lines and Open Source.* Dagstuhl, Germany, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.

Van Gurp, J. (2006).'OSS Product Family Engineering', in *1st International Workshop on Open Source Software and Product Lines*, Baltimore, Maryland, USA, 22nd August,

Vidyasagar, P. & E. Chang (2004).'Open Source and Closed Source Software Development Methodologies', in *4th Workshop on Open Source Software Engineering*, Edinburgh, Scotland, May,

Wesselius, J. (2008) 'The Bazaar inside the Cathedral:Business Models for Internal Markets', *IEEE Software*, 25 (3)**,** 60-66.

Butler, S., D. Adebanjo and H. Ismail (2008) "Open Source Software and Leveraging of Business Effectiveness in Smes - a Case Study", *Ice-B 2008: Proceedings of the International Conference on E-Business,* pp. 93-100.

Capra, E. and A. I. Wasserman (2008) "A Framework for Evaluating Managerial Styles in Open Source Projects", *Open Source Development, Communities and Quality,* **275** pp. 1-14.

Castells, M. (1996) *The Rise of the Network Society, the Information Age: Economy, Society and Culture,* Blackwell, Oxford.

Castells, M. (2001) *The Internet Galaxy: Reflections on the Internet, Business and Society,* Oxford University Press, Oxford.

Demil, B. and X. Lecocq (2006) "Neither Market nor Hierarchy nor Network: The Emergence of Bazaar Governance", *Organization Studies,* **27 (10),** pp. 1447-1466.

Fitzgerald, B. (2006) "The Transformation of Open Source Software", *MIS Quarterly,* **30 (3),** pp. 587-598.

Fogel, K. (2005) *Producing Open Source Software: How to Run a Successful Free Software Project,* O'Reilly, Sebastopol, CA.

Hyatt, J. and M. Mickos (2008) "The Oh-So-Practical Magic of Open-Source Innovation", *Mit Sloan Management Review,* **50 (1),** pp. 15-19.

Koponen, T., H. Lintula and V. Hotti (2006) "Defects Reports in Open Source Software Maintenance Process - Openoffice.Org Case Study", *Proceedings of the 10th IASTED International Conference on Software Engineering and Applications,* pp. 434-439.

Lindman, J., M. Rossi and P. Marttiin (2008) "Applying Open Source Development Practices inside a Company", *Open Source Development, Communities and Quality,* **275** pp. 381-387.

Ljungberg, J. (2000) "Open Source Movements as a Model for Organizing", *European Journal of Information Systems,* **9 (4),** pp. 208-216.

Melian, C. and M. Mahring (2008) "Lost and Gained in Translation: Adoption of Open Source Software Development at Hewlett-Packard", *Open Source Development, Communities and Quality,* **275** pp. 93-104.

O'Mahony, S. and F. Ferraro (2007) "The Emergence of Governance in an Open Source Community", *Academy of Management Journal,* **50** pp. 1079-1106.

O'Reilly, T. (1999) "Lessons from Open Source Software Development", *Communications of the ACM,* **42 (4),** pp. 33-37.

Pulkkinen, M., O. Mazhelis, P. Marttiin and J. Meriluoto (2007) "Support for Knowledge and Innovations in Software Development - Community within Company: Inner Source Environment", *WEBIST 2007: Proceedings of the Third International Conference on Web Information Systems and Technologies, Vol SeBeG/eL,* pp. 141-150.

Raymond, E. (1999) *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary,* O'Reilly & Associates, Sebastopol, California.

Shah, S. K. (2006) "Motivation, Governance, and the Viability of Hybrid Forms in Open Source Software Development", *Management Science,* **52 (7),** pp. 1000-1014.

Shaikh, M. and T. Cornford (2008) "Open-Sourcing: On the Road to the Ultimate Global Source?" in *2nd Global Sourcing, Services, Knowledge and Innovation Workshop* Val d'Isere, France, March 2008,
Shaikh, M. and T. Cornford (2009) "Managerial Strategies for Open-Sourcing Innovation". in *Third Global Sourcing Workshop: The Impacts of Global IS Sourcing on Engineering, Technology and Innovation Management,* Keystone, CO, USA, 22-25 March 2009,

Välimäki, M. (2003) "Dual Licensing in Open Source Software Industry", *Systemes d´Information et Management,* **8 (1),** p. 63+.

Ven, K. and H. Mannaert (2008) "Challenges and Strategies in the Use of Open Source Software by Independent Software Vendors", *Information and Software Technology,* **50 (9-10),** pp. 991-1002.