



An Empirical Investigation into the Adoption of Inner Source in IT Companies: A Case Study

Master's Thesis in Software Engineering and Management

NILOOFAR SAFAVI

An Empirical Investigation into the Adoption of Inner Source in IT Companies: A Case study

NILOOFAR SAFAVI



UNIVERSITY OF
GOTHENBURG



CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering

Chalmers University of Technology

University of Gothenburg

Göteborg, Sweden 2019

An Empirical Investigation into the Adoption of Inner Source in IT Companies: A Case Study

NILOOFAR SAFAVI

© NILOOFAR SAFAVI, 2019

Supervisor: Imed Hammouda

Examiner: Eric Knaus

Master's Thesis 2019

Department of Computer Science and Engineering

Chalmers University of Technology

University of Gothenburg

SE-412 96 Göteborg

Sweden

Telephone + 46 (0)31-772 1000

Cover:

Inner Source in a corporation [51]

An Empirical Investigation into the Adoption of Inner Source in IT Companies: A Case Study

NILOOFAR SAFAVI

Department of Computer Science and Engineering

Chalmers University of Technology and University of Gothenburg

Abstract

Inner Source is a rather new concept and introducing it to companies involves some challenges. In this thesis, we investigate the challenges and obstacles of adopting Inner Source in a large IT company. The results are then analyzed and summarized.

In addition, the company owns many products and needs to decide which products are suitable for inner sourcing purpose. The criteria for selection of the products are investigated and the results are compared to the results of the previous studies.

In the final stage, we investigate a framework for adoption of Inner Source tailored to the needs of the company and compared the results to other available frameworks.

Keywords: Inner Source, adoption, challenges, criteria, framework

Acknowledgments

I would like to thank Imed Hammouda, my supervisor at the university who helped me setup the project and showed me the directions to perform an industrial project using academic approaches.

I am also especially thankful to Eric Knaus, my examiner for his valuable feedbacks.

Finally, I would like to thank all interviewees and members of the studied projects who helped me throughout the study.

Niloofar Safavi

Table of Contents

1. Introduction.....	1
1.1. Statement of the problem.....	1
1.2. Research Questions	2
1.3. Aims and objectives.....	2
1.4. Scope	2
1.5. Definitions, acronyms and abbreviations.....	3
1.6. Main contributions	3
1.7. Structure of the thesis	3
2. Background and related work	4
2.1. Open Source.....	4
2.2. Inner Source	5
2.3. Inner Source challenges	6
2.4. Inner Source product	7
2.5. Inner Source implementation.....	8
3. Research methodology.....	14
3.1. Design and planning	15
3.2. Preparation for data collection.....	15
3.3. Collecting evidence.....	15
3.4. Analysis.....	18
3.5. Reporting	18
4. Results.....	19
4.1. Challenges.....	19
4.2. Product	23
4.3. Adopting Inner Source by the company.....	27
5. Discussion.....	38
5.1. Challenges.....	38
5.2. Product	39
5.3. Adoption model	40
6. Conclusions.....	46
6.1. Threats to validity.....	46

6.2.	Implications for research	47
6.3.	Implications for practitioners	47
7.	<i>References</i>	48
8.	<i>Appendixes</i>	52
8.1.	Appendix 1: Successful Open source projects	52
8.2.	Appendix 2: Questionnaire (Guideline for interviews).....	54
8.3.	Appendix 3: Candidate products for inner sourcing.....	56

List of Figures

Figure 1-Sharma et. al framework [55]	10
Figure 2- Key factors for adopting Inner Source [17]	13
Figure 3- Interviewees familiarity with OS	17
Figure 4- Interviewees familiarity with Inner Source	17
Figure 5-Current products status	20
Figure 6-Company software	23
Figure 7 - How Gerrit works [47]	29
Figure 8- Summary of Inner Source Challenges at the company	38
Figure 9- Adoption model at the company	41
Figure 10- Adoption framework by the company	42

List of Tables

Table 1- Inner Source challenges (Morgan et al. [24])	6
Table 2-Inner Source challenges by Stol et al. [52]	6
Table 3-Differences between Infrastructure based and Project-based Inner Source [18][52].....	9
Table 4- Sharma et. al framework sub categories [55]	11
Table 5- Stol et al. framework [52].....	12
Table 6- Interviewees roles	16
Table 7- Product criteria according to our study vs. other studies	40
Table 8- Comparison between our study and Torkar et al. [56]	43
Table 9-comparison between my study and Stol et al. [18] study	44
Table 10- Relation between the proposed framework and reported challenges.....	45

1. Introduction

Open-source software (OSS) is software with its source code which is available with a license that allows everyone to download, use and change the software [4]. The source code is not hidden from users and they are considered as beta testers. Today there are many successful examples of OSS available such as Linux, Apache server, Mozilla, etc. It is reported that adopting open source software resulted in saving of 60 billion dollars per year [27]. Besides, adopting OSS has many benefits including low cost, increased quality, involvement of skilled developers from all around the world which means more people are monitoring the project, learning from the community, etc. On the other hand, it involves many challenges such as documentation, support and maintenance, integration, migration, etc. [28].

Applying Open Source Software development practices and tools within a company is called Inner Source [18]. The resulting product in Inner Source is proprietary unlike open source which comes with open source license [38].

Inner sourcing is becoming more and more popular. Gaughan et al. [42] explains that the motivations for adopting this approach is the possibility to get support from distributed developers and increase of software reuse and quality as a result of increased software availability and transparency. In addition, everyone in the organization is invited to participate either as developer or contributor which could lead to innovation and increased awareness plus increased speed of development. However, like open source software development, it is not a straightforward approach. It has many effects on the organization. It affects the way projects are sourced and managed. In addition, the adoption model must match the organization goals and scope. It also affects the operational aspects such as how developers across the company can access and modify the source code [42]. Therefore, companies need to study it carefully before deciding to go for it.

1.1. Statement of the problem

The study is conducted in a large company working with software development and also a large IT service provider. The company benefits from several open source software available and as inner-sourcing is becoming more and more popular, the company would like to investigate the possibility of adopting inner-sourcing within the company. There are many motivations behind this. One is that many software products are developed in different organizations but due to the large size of the company, it is impossible to know whether a required tool or library has ever been created by another team or not and therefore there is a high chance that different departments reinvent the wheel many times which is in fact very costly for the company. Another problem is that sometimes projects are being paused or delayed due to lack of resources or knowledge within a department. Outsourcing is not the preferred option as it is costly. Having the possibility to involve developers and experts from other teams will eliminate this problem. Furthermore, several other benefits mentioned in the studies on companies which use Inner Source motivated the company to investigate the Inner Source adoption possibility.

However, unlike Scrum which is a methodology with certain guidelines and roles, inner-sourcing is not a methodology and is a set of practices that needs to be customized to the needs of each organization [17]. It must suit the context and structure of the organization. There are lots of questions that need to be answered. First, we need to understand the state of the art in inner sourcing and how Inner

Source could be implemented and what are the problems and challenges that lies ahead in order to address them properly. The purpose of this study is to find the answers to these questions.

From the research perspective, little study has been done on how organizations should implement Inner Source or assess their possibility of adopting Inner Source and each of those studies are limited in one or more ways. Furthermore, several successful Inner Source implementations have been reported but little investigation has been done on the obstacles and challenges practitioners encounter for ISS adoption [17].

1.2. Research Questions

- **RQ1: What are the obstacles and challenges of adopting Inner Source?**

Open source development practices are different from traditional software development practices used by companies. In addition, inner-sourcing involves engagement of different people from different teams [16]. This new way of working might introduce new challenges. We are going to investigate the challenges for adopting Inner Source by the company.

- **RQ2: Which software components/products are best candidate for inner-sourcing?**

Not all packages are a good choice for inner-sourcing. We need to identify which ones are more suitable for inner-sourcing through identifying the criteria that helps companies pick the right candidate. The findings of this section help practitioners specify the requirements for the Inner Source product.

- **RQ3: How can the company adopt Inner Source?**

The company uses traditional software development methods, but inner-sourcing is a new philosophy and requires a different approach. New ways of working, tools and practices, quality assurance, management of the product, communication and collaboration with the community are subjects to think about when opting to go for Inner Source. Furthermore, the models of adoption of Inner Source vary from organization to organization and needs to be tailored to suit the needs of each organization [16]. We investigate the practices and methods suggested by the professionals in the company and compare the results to the available frameworks proposed by other studies. In addition, we would like to see whether it is possible to address the challenges found in RQ1 using the framework.

1.3. Aims and objectives

The main purpose of this master thesis is to investigate the possibility of adopting inner-source within the company. This, thus, implies:

- Understanding state of the art of Inner Source.
- Identifying the main challenges and obstacles for adopting Inner Source
- Reviewing and studying products and the processes being used inside the company in order to identify the best candidates for inner sourcing.
- Suggesting a guideline to implement Inner Source at the company

1.4. Scope

The scope of this study are major software development divisions in the company. Software development is carried out in different locations including Bangalore-India, Wroclaw-Poland and

Gothenburg-Sweden. Major development tracks are Java, .Net, JavaScript and mobile application (IOS and Android). The interviewees were selected from the above-mentioned development teams.

The focus of this study is Inner Source and Open source is not covered. In addition, the focus is only on internal products and customer products are excluded since different regulations and procedures are applicable to them.

1.5. Definitions, acronyms and abbreviations

- OSS: Open source software
- OSSL: Open source software license
- OSI: Open source initiative
- ISS: Inner Source software
- POS: Progressive open source

1.6. Main contributions

Most of the existing literatures on Inner Source are case studies which discuss the benefits Inner Source has brought to organizations. Little work has been done on investigating the related challenges in the actual use context. In addition, there are few studies which provide a general framework for Inner Source implementation and validate their findings.

The main contributions of this thesis to identify the challenges companies face for adopting Inner Source and introduce a framework which helps organizations overcome the challenges. Identifying the criteria for selecting the inners source product is another contribution of this thesis. Areas that require further research are also identified.

1.7. Structure of the thesis

- Section 1: Introduction to the subject. Problem statement and research questions are presented here.
- Section 2: Background and related work. Previous studies related to our research questions are investigated. Our purpose is to find out if previous work done in this field can be used for current study and if we could benefit from successful experiments and help us find answer to our questions.
- Section 3: Research methodology and data collection methods are described.
- Section 4: Study results. We focus on the case study results and the process followed in the company.
- Section 5: Discussion and analysis of the results of the theoretical and empirical research.
- Results of the theoretical and empirical research are compared
- Section 6: Conclusion is used to discuss the most important results and evaluation of the method and process used in this study. It also provides implications for further research.
- Section 7: References
- Section 8: Appendixes

2. Background and related work

2.1. Open Source

As software systems grew in complexity, traditional development methods became inefficient. As a result, lots of efforts have been put into developing new methods and techniques to design and implement software systems and numerous methods were proposed: from the traditional waterfall technique, Iteration model, V-shaped model, Spiral model, Extreme model to recently developed and highly popular agile methods [2].

These methods aim to provide a structured and systematic approach to develop software and lower the costs by making the development processes lighter weight, faster, nimbler and more reliable [3]. While there have been lots of debates as which process is the best and lots of research have been carried out to compare these techniques and bring to light their different aspects, the emergence of a phenomenon called “Open Source Software Development” and its achievements and successes has been astonishing.

“Open Source Software (OSS) is software distributed with a license allowing access to its source code, free redistribution, the creation of derived works and unrestricted use.” [4] This means there is no limitation and restriction in using the software or its source code as long as the criteria listed in the license are met. One might only use the software like other closed source software while another might review the code out of curiosity or to fix a bug. A developer or an organization might further develop the source code to meet their needs, etc. Open source software licenses “OSSL”, vary from one to another but they all have some criteria in common which are the fundamentals of OSSL. “The source must be available for redistribution without restriction and without charge, and the license must permit the creation of modifications and derivative works and must allow those derivatives to be redistributed under the same terms as the original work [7]. The most important purpose of licensing in this type of software is to deny anyone the right to exclusively exploit them which is exactly the opposite of what it is in other types of software [5].

Today, OSS development method is receiving more and more attentions and poses a very genuine challenge to commercial software business and the organizations producing them [6]. Many companies have switched from traditional development methods trying to win back market share, increase product growth and improve market penetration [8]. There are also lots of successful and well-known projects and products being produced using OSS development method such as Linux, UBUNTU, BSD, MySQL, Apache, Firefox, WordPress, BIND, Perl, SendMail, etc. (For more detailed information about these products, see Appendix 1).

2.1.1. Benefits of OSS development

It has been claimed that OSS development creates opportunities for a system to be developed much more quickly [6][8][10]. There are usually many developers all around the world who are willing to collaborate on open source projects which enables the products to be developed much faster than the products produced using other methods.

Creativity is another success factor that plays a very important role in OSS development. It has been reported that open source projects boost the creativity and most likely will result in chart topping products that increases the market share and business revenue [6][7][10].

It is also claimed that open source products are recognized for their high level of quality including reliability, efficiency and robustness [12]. Fewer number of defects compared to other products as a result of the peer reviews by a large number of talented developers which leads to the defects to be detected and fixed more rapidly [6] [12] [13].

2.2. Inner Source

The astonishing success of OSS projects has been an inspiration to companies to benefit from open source development principles internally [16]. The process of adopting OSS development inside a company is called Inner Source [17]. Stol et al. defines Inner Source as follows: “The leveraging of Open Source Software development practices within the confines of a corporate environment”. As such, it refers to the process of developing software at an organization that has adopted OSS development practices.” The main difference between these two practices is that in Inner Source, the source code is only accessible inside the organizations boundaries as opposed to OSS development which provides access to the source code globally [18].

2.2.1. Benefits of Inner Source

Since inheriting most of the characteristics from OSS development, the benefits of adopting Inner Source inside a company are quite similar to those of OSS projects. The reasons for adopting Inner Source might vary from one company to another. However, some of the most important motivations behind inner sourcing are discussed in this section.

- Improve in code reuse: having all the projects shared in an internal repository gives everyone inside the organization the opportunity to reuse the work done by others and prevent unnecessary rework which is obviously a waste of time and resources [19].
- Improved quality: having the contributions being peer reviewed by a bigger community of developers results in the defects to be detected and fixed more efficiently. This might also result in contributors to write good codes due to the contributor’s reputation being under scrutiny at organizations level [16] [20].
- Quick developer redeployment: since developers are familiar with the projects shared on the Inner Source repository as well as infrastructures and common tools inside the organization, they can more easily and quickly be reassigned to other projects which subsequently result in time to market and project start up time to be reduced significantly [16] [21].
- Increased awareness: an open environment inside an organization increase awareness between different departments involved in the projects from development, to maintenance, support and management which improves the efficiency and prevents duplicate works [16] [22] [23].
- Larger developers pool: open collaboration inside a company results in a larger number of developers which broadens innovation and increases the overall expertise level [16] [20] [23] [25].
- Increased development speed: collaboration between development teams might result in an increase in overall efficiency and development speed as a single team may not have enough capacity to implement all the requirements in time [16] [21] [25].

2.3. Inner Source challenges

Previous case studies reported several challenges faced by practitioners. In this section, we point out some of them:

Morgan et al. identified a list of challenges associated with inner sourcing networks as shown in Table 1. [24]

Challenges	Explanation
Achieving a high level of commitment	Commitment by all stakeholders is crucial but difficult to achieve. Product team leaders are often only concerned with having work on their project done correctly and are not overly concerned about the platform. People would be more committed and motivated if incentives were offered.
Increasing knowledge sharing and exchange	Difficult to get people to invest time or effort in sharing and building skills and knowledge outside their own domain.
Achieving a common vision	Important to have a clear unifying vision so people don't waiver off in different directions and send out mixed messages to the rest of the teams
Effective governance of the network	Managers are used to controlling and regulating things in a closed environment and do not offer incentives that encourage commitment, transparency and knowledge exchange in an informal capacity in the network.

Table 1- Inner Source challenges (Morgan et al. [24])

Stol et al. [52] identified thirteen challenges of adopting Inner Source and compared them to the challenges of open source adoption. The challenges were divided to four categories:

- Documentation
- Community support and maintenance
- Integration and architecture
- Migration and usage [52]

The complete list is displayed in Table 2.

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

Table 2- Inner Source challenges by Stol et al. [52]

Dinkelacker et al. investigated the organizational and infrastructure challenges of implementing Progressive Open Source (POS) at HP. [21]

Gurbani et al. briefly points out some challenges of inner sourcing. He mentions tool compatibility issues and conflicts in building strategies of groups as the challenges they faced at Alcatel Lucent. [49]

Malin et al. [48] found similar organizational challenges in their study on building networks of software communities in large corporations which investigates POS at HP. According to the above studies, organization challenges include:

- **Virtual Organization**

Most companies have a hierarchical organization structure. Virtual organizations are people who work beyond constraints and organizational boundaries and it is a challenge to incorporate that into time critical business market which each division and manager has their own roadmaps.

- **Leadership**

In Inner Source, leader is the main owner of the shared asset which many other products are dependent on it. The company faces a big challenge if the leader decides to leave the company and no proper replacement is found.

- **Task assignment**

In open source people pick up tasks voluntarily and according to their skillset and the resources are huge number of developers who are willing to participate from around the world. This is not the case for companies as resources are limited to its employees. In addition, manager of one department cannot assign tasks to employees of other departments [21] [48].

2.4. Inner Source product

Identifying the suitable product for inner sourcing is a very important part of the process. To achieve this, first the criteria for product needs to be specified and then based on those criteria appropriate products can be selected.

Not many studies have focused on this topic, but a few share their learned lessons from the case studies carried out in different companies. One of them is Stol et al. reported the results of three case studies done in three large organizations including Philips Healthcare, Neopost and Rolls-Royce. They suggest that the start point should be a seed product that has a significant business value to the company. The product should be an existing one which is runnable. The reason is that starting an Inner Source project from scratch and expecting to find contributors is quite difficult [50]. Raymond states that one cannot code from scratch in bazaar style. It is possible to debug, test and enhance the code but originating a project from scratch and expecting to attract developers is quite hard since developers need to have something to play with [11]. Wesselius claims that the seed product must have clear specifications so that it can be outsourced to other development teams [25]. Senyard and Michlmayr claim that the initial phase known as “Cathedral” phase must be implemented by a small group. This could act as a base for further development in the Bazar Style [1].

On the other hand, Gurbani et al. hypothesis was that lacking a seed product, research or technology is a very good start point [49] [17].

In addition, Stol et al. suggest the product should have a modular architecture. Basically, modularity is a desired aspect for software but it of high importance when it comes to Inner Source because it enables developers to work on different modules simultaneously [17].

2.5. Inner Source implementation

Inner Source is derived from open source which is based on the principle of meritocracy [53]. Meritocracy is a general term for the following more specific principles of open source:

- **Egalitarian:**

Projects are open on the internet for anyone who would like to contribute.

- **Meritocratic:**

Contributions are not evaluated by people's experience or roles in the organization but by how good they are. Decisions are discussed publicly and can be looked up for reference.

- **Self-organizing:**

Community is self-organizing and there is no imposed process which determines how people should work in a project.

These principles contrast with how most companies manage their software development processes as stated below:

- **Assigned jobs:**

Tasks are assigned from higher managers and they decide who should work on what project and software. Volunteer contribution is uncommon.

- **Status rather than merit:**

The status of people in the organization hierarchy determines who has the final word in making final design and implementation decisions.

- **Imposed processes:**

The processes and tools to be used in software development is pre-defined and it must be followed by all projects [54].

2.5.1. Adoption models

According to Feller and Fitzgerald there is no single Open Source software development process, and just many open source projects which keep reoccurring [39]. Although OSS projects follow different practices, all share a common philosophy [40]. This raise the question what adopting OSS means for a company when there is no commonly accepted definition of Inner Source [18].

Gaughan et al. studied seven cases of Inner Source adoption and found that each case was a 'unique' implementation of Inner Source [18][42].

However, Stol et al. conducted a literature review and identified a few common practices in Inner Source such as distributed development [43], open development practices such as peer-review [25]

[42], availability of the source code and the possibility of contribution by anyone inside the company [21][18].

Two implementation models of Inner Source are suggested by Gurbani et al. [44]:

- Infrastructure-based Inner Source model
- Project-based Inner Source model

2.5.1.1. Infrastructure based Inner Source model

In this model, the company provides the required infrastructure such as tools, mailing lists and code repositories and developers use them to host their software development projects [18][44]. This model was applied by SAP [45] and also at Nokia as “iSource” initiative [46]. It was also applied at HP in the Progressive Open source initiative [21].

Sharing of software is increased using this model but reuse remains ad-hoc. Furthermore, support is optional and very much dependent on the individual project [17].

2.5.1.2. Project-based Inner Source model

In this model a division which is called core team takes care of the common code known as shared assets and handles the inner sourcing activities such as making the code available to other units and handling the support and maintenance of it. This model was applied at Alcatel Lucent [44] and Philips healthcare [25]. Reuse in this model is the main goal and is planned. Support of the shared asset is the responsibility of the core team.

2.5.1.3. Differences between Infrastructure based and Project-based Inner Source model

A summary of the main differences between the two models are displayed in Table 3.

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

Table 3-Differences between Infrastructure based and Project-based Inner Source [18][52]

2.5.2. Available frameworks for Inner Source

Many researchers published guidelines and lessons learnt from adoption of OSS practices in different companies [17][21][49][50]. In this section, the proposed frameworks are briefly described.

2.5.2.1. Sharma et al.

Sharma et al. proposed a framework for creating hybrid-OSS communities in the companies as shown in Figure 1. The framework consists of three major elements: community building, community governance and community infrastructure. [55]

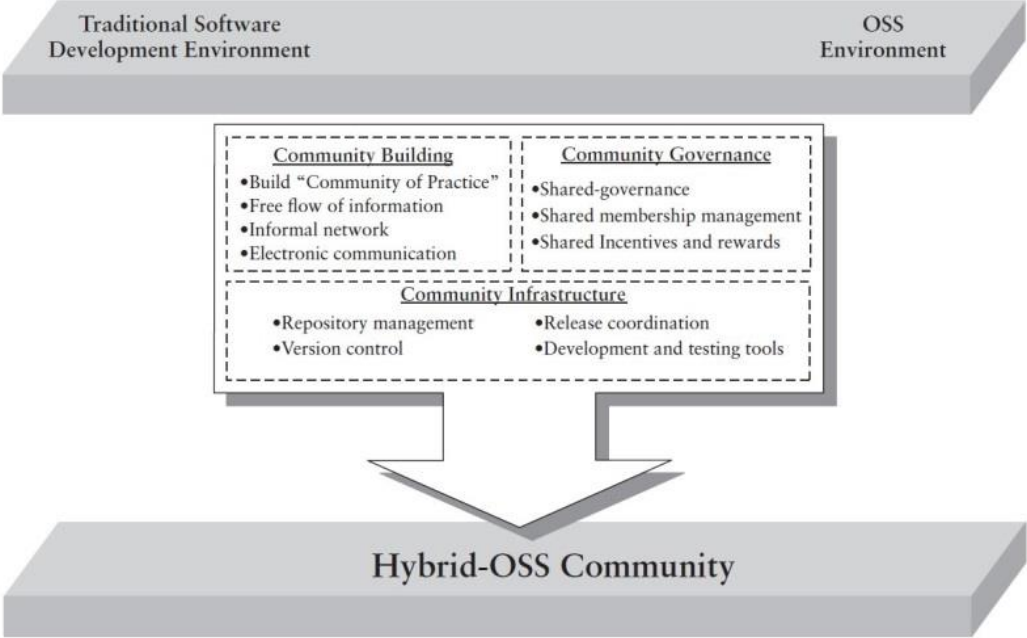


Figure 1-Sharma et. al framework [55]

Each category is divided into subcategories which are described briefly in Table 4.

Element	ID	Activity	Description
Community building	F1	Build “community of practice”	Promotion of free exchange of ideas and information among their workers.
	F2	Free flow of information	Allow workers to leverage the knowledge of others and identify potential opportunities for new innovations.
	F3	Informal network	Get rid of high degree of formal structure and provide mechanisms for workers to complete tasks through informal relationships and networking.
	F4	Electronic communication	Encourage electronic communication among groups across locations
Community governance	F5	Shared-governance	1. Manage community of practice in a way that is perceived as fair and equitable, in order to sustain motivation. 2. Move away from imposing central command and control structure on the community. 3. Allow community members to work in teams and empower them to make decisions by discussion and voting.
	F6	Shared membership management	4. Provide mechanisms for qualified people to join the community and contribute to the project. 5. Allow new members to join and current members to leave. 6. Allow members to forge and dissolve relationships with outside entities (e.g., customers, suppliers, vendors) as they see fit.
	F7	Shared incentives and rewards	Develop a performance and measurement system, which rewards and promotes members based on meeting both community and organisational goals.
Community infrastructure	F8	Repository management	Provide infrastructure to store and manage artefacts.
	F9	Version control	Before committing changes to artefacts (e.g., source code) they need to be evaluated by peers for quality and generality.
	F10	Release coordination	Mechanisms for product release and documentation must be in place.
	F11	Development and testing tools	Provide necessary tools and infrastructure for software development and project management.

Table 4- Sharma et. al framework sub categories [55]

2.5.2.2. Torkar et al.

The following recommendations are made by Torkar et al. [56] as Inner Source adoption opportunities:

- Reduce technical debt
- Define an entry path for newcomers
- Increase information availability and visibility
- Embrace asynchronous tools for communication and decision making
- Let developers influence ways of working
- Allow task selection [56]

2.5.2.3. Stol et al.

Stol et al. performed a comprehensive literature review of the available frameworks for inner sourcing. They claimed that other frameworks focus on few aspects of Inner Source adoption therefore they proposed a new framework according to Table 5 which comprised of 19 factors divided to 4 main categories to assess the organization compatibility for inner sourcing [52].

Category	ID	Element	Description
Software product	SP1	Runnable software product	There should be an initial set of components or product that is runnable and usable.
	SP2	Needed by several stakeholders	Need to pool resources; contributions are useful to everyone. Support diversity of usage and encourage plurality of authorship; reusable in different contexts
	SP3	Maturity state	Requirements and features not fully known at the outset. Need to evolve continuously.
	SP4	Utility v. simplicity	Balance between a component's functionality and simplicity.
	SP5	Modularity of software product	Should be highly modular to enable parallel development of the code.
Development practices	DP1	Requirements elicitation	Requirements are asserted after the fact
	DP2	Implementation & quality control	Truly independent peer review
	DP3	Release management	Release early, release often
	DP4	Maintenance	Maintenance as reinvention
Tools & Infrastructure	TI1	Development tools	Common tools
	TI2	Infrastructure for open access	Infrastructure to facilitate open access to everybody in organisation. Wiki.
Organisation & Community	OC1	Work coordination	Task selection v. assignment
	OC2	Communication	Electronic and asynchronous communication
	OC3	Leadership & decision making	Core team; reputation and meritocracy; trusted lieutenant
	OC4	Motivation & incentives	Start small, demonstrate value; motivation; facilitate change
	OC5	Open development culture	Promote an open development culture.
	OC6	Organisational support	Need to have backing from management

Table 5- Stol et al. framework [52]

In a later study Stol et al. [17] updated the framework. They identified 9 key factors for adopting Inner Source and divided them into three categories as shown in Figure 2.

- Software product
- Practices and tool
- Organization and community

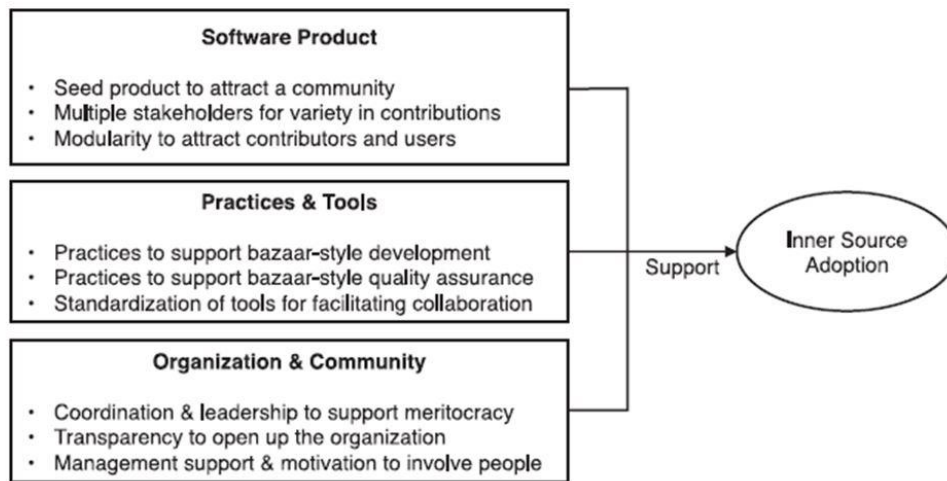


Figure 2- Key factors for adopting Inner Source [17]

3. Research methodology

Generally, there are two types of research; Quantitative and Qualitative. Lichtman describes quantitative researches as gathering numerical data in order to examine and test hypothesis [29].

While gathering statistical data and examining trends are very useful, they don't tell the whole story. Questions such as what caused such improvements or deteriorations, how did they happen or why they happened remain unanswered. These types of questions can only be answered by data obtained during careful observations [29]. This results in a type of studies called qualitative research which is gathering qualitative data such as open-ended responses, interviews, participant observations, field notes or points of views in order to understand and interpret social interactions [30] [31].

The major differences between these two approaches are as follows:

- Qualitative research uses open-ended questions which are more flexible and gives the participants the freedom to elaborate their responses compared to fixed and non-flexible questions of quantitative method that usually uses a structured and fixed set of questions and predefined choices like "yes" or "no" or something similar [32].
- Qualitative research collects textual data that is obtained from video or audio tapes or personal notes while quantitative research collects numerical data that is obtained by assigning numerical values to the responses [32].
- the objective of qualitative research is to obtain detailed understanding of reasons and motivations to find answers to questions such as "what", "why", "how" and also the underlying information about the processes compared to quantitative research where the objective is to quantify a research problem by answering questions such as "how much" and "how often" and then to generalize it to a larger population [33].
- Quantitative research needs large number of respondents to achieve valid findings and the ability for generalization while qualitative research only investigates a small number of participants [33].
- Qualitative and quantitative methods use different data collection techniques and analyze them differently. Quantitative research usually uses surveys or questionnaires and the results will be presented in form of statistical trends and patterns while qualitative research uses interviews or focus group discussion to understand the behaviors and the processes by interpreting the views and experiences of the participants [33].

Due to the nature of the problem under study, a qualitative research methodology was selected because as described in section 1.1 and 1.2, the purpose of this study is to extract the facts about the current state of the Inner Source inside the company and obtain understanding about its underlying reasons, opinions and motivations. The aim is to gain insight into the problem and develop ideas, suggestions and solutions. This led the study to be conducted using a qualitative research approach which perfectly suited the needs of this work.

Having examined different qualitative research techniques, the case study was found to be the best choice and could efficiently meet the needs of the study. According to Runeson and Höst [34], the case study methodology is well suited for software engineering researches where the objects of study are contemporary phenomena, which are hard to study in isolation. Schell describes the case study as "The most flexible of all research designs, allowing the researcher to retain the holistic characteristics of real-life events while investigating empirical events" [35]. A case study is an empirical enquiry that

investigates a contemporary phenomenon within its real-life context especially when the boundaries between phenomenon and the context are not clearly evident and the data must be obtained from multiple sources [35] as was the case in this study.

According to Runeson and Höst, a case study comprises five major steps including design and planning, preparation for data collection, collecting evidence, Analysis and Report which are described in the following sections [34].

3.1. Design and planning

Like any other task, a case study needs to be designed and planned well ahead of the study begins. To do this a list of the subtasks that needed to be carried out during the project was created. According to Neale et al. [36] the most important questions that need to be answered are as follows:

- What are the objectives of the study?
- Who are the stakeholders involved in the study?
- How are they going to contribute?
- What is the necessary information that needs to be extracted?
- What are their potential sources?

A list of the tasks was created and planned accordingly.

3.2. Preparation for data collection

In this step, data collection techniques were selected and the overall instruction on how to collect data was specified as below:

- **Interviews:**

Interview questions were prepared, and invitation was sent to interviewees. A guideline was created for interviewees to get an overview of the topic prior to meeting. Interview questions were reviewed by my supervisor to ensure they cover the topic and fulfill the requirements of the thesis. Recording method was selected and finally meeting invitation was sent to respondents.

- **Literature review:**

Academic libraries to be searched were identified and keywords for search were specified.

- **Workshop:**

Neale et al. describes that a brainstorming is critical in this stage to bring into light different aspects of the study [36]. Workshops were scheduled with different teams to discuss the findings on research question 2 regarding product criteria for inner sourcing and based on that the teams identified software products candidates for inner sourcing.

3.3. Collecting evidence

Data collection was performed using literature reviews, interviews and workshops.

3.3.1. Literature review

A semi-structured literature review was carried out in the study. The purpose was to find relevant articles on Inner Source to understand the concept and its application and practices.

First, the term “Inner Source” was used to search the article titles in Google Scholar which led to identification of 10 records out of which only 2 were relevant. Having scammed some articles, it turned out that “Corporate open source” and “Progressive open source” and “Internal open source” are used interchangeably. We combined those terms and the keyword (“Corporate open source”) OR (“Inner Source”) OR (“Internal open source”) OR (“Progressive open source”)) was formed. This keyword was then used to search in Meta data for articles in digital libraries including IEEE Xplore, ACM digital library, Science Direct and Springer. We limited our search to papers in computer science and software engineering domain with no limit on publishes date. This led to identification of 36 papers.

Literature review was carried out later to search for the challenges and obstacles that software engineers face in inner sourcing. An “in title” and “in abstract” search was carried out in the aforementioned libraries. First the term “Inner Source challenges” was used. Then a search for articles on “Inner Source product” was carried to compare our study results with other case studies. Finally, we used the term “Inner Source framework” and “Inner Source adoption model” to find out if there are any frameworks proposed for inner sourcing by other studies. 25 relevant articles were found based on the above key words.

3.3.2. Interviews

Interviews were used to collect qualitative data. The first step was identifying the teams who either have experience with Inner Source or work with software development and the products which have the potential to be Inner Sourced implying that they can be used as shared assets in the company.

After extensive search five main development teams were identified. 16 people were selected with different roles to cover different aspects of the software development process and to get a broad view of the subject. The interviewees’ roles are listed in Table 6.

Role	Number
Project Manager	4
Software Architect	6
System/business Analyst	2
Developer	4

Table 6- Interviewees roles

The interview focused on different aspects of Inner-sourcing. Since Inner Sourcing is very similar to Open Source (OS) prior understanding of it is helpful. Interviewees prior experience in open source is displayed in the Figure 3. As can be seen everyone had a prior knowledge of OS concept. Only two people had experience contributing to open source development and others had a basic knowledge of OS.

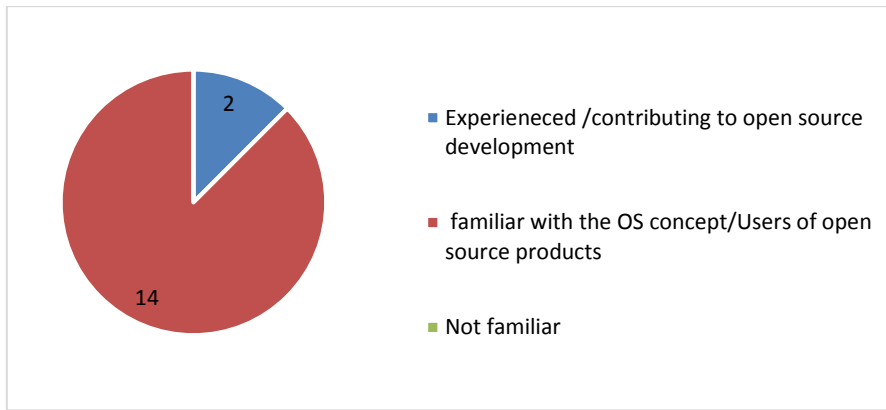


Figure 3- Interviewees familiarity with OS

The company has previous experience with inner sourcing and the interviews are selected both from department with inner sourcing experience as well as departments which have not tried it but are willing to adopt Inner Source. The distribution of the interviewees is as Figure 4.

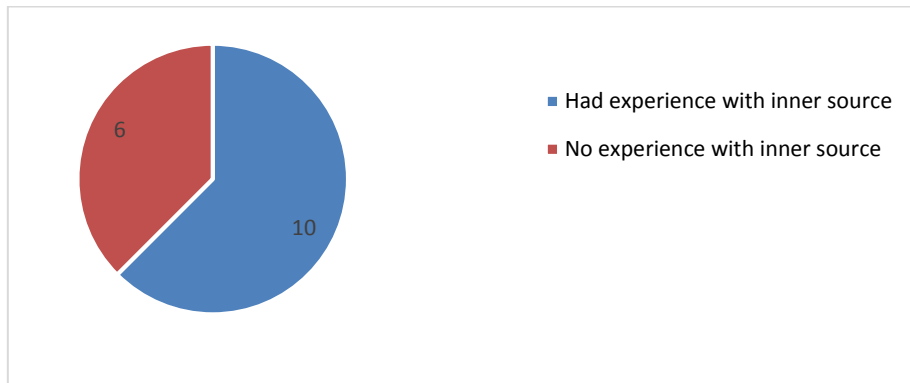


Figure 4- Interviewees familiarity with Inner Source

Meeting request was sent to interviewees explaining the purpose and goals of the thesis and the interview. A brief introduction text was sent to the interviewees prior to the meeting so they can prepare and get familiar with the topic in case they were not already.

Meetings were carried out in the forms of online meetings and face-to-face meetings depending on the geographical location of the interviewees. The sessions were around one hour, and the interviews occurred during a two-month period.

An open-ended questionnaire with 20 questions under three main categories was used to conduct the semi-structured interviews. Further questions were asked followed by the predefined questions if needed. An interview guideline was used during the interview which helped me ensure all my questions were covered in the interview and it worked as a memory aid too. The interviews were recorded and then transcribed. Notes were also taken during interviews.

3.3.3. Workshops

Three workshops were set up with main teams as a means of brainstorming to identify the products which are suitable for inner sourcing based on the criteria identified during interviews. In addition,

adoption models were discussed during the workshops and the results of the interviews were discussed and confirmed by participants.

3.3.4. Ethical concerns

The study purpose and data collection method were discussed in advance with participants and the related managers. The participation was voluntary, and interviews were recorded with the permission from the interviewees and then transcribed. The interviewee identities were kept anonymous while transcribing. The outcome was sent to interviewees for confirmation in order to preventing misunderstandings. The results of the analysis were also sent to participants for reviewing.

3.4. Analysis

Recordings, notes and transcriptions from the interviews were collected and used to summarize the results. The interviewee answers are presented in a table using keywords. After simplifying and summarizing, we will be able to find out the common opinions and use it to perform a root cause analysis. Graphs are used to present the analysis results.

Later the results of the literature review and the previous studies in this field are compared to the results of this case study to find out whether our study confirms the previous research results or not.

3.5. Reporting

The last step was reporting the findings. A draft was prepared and sent to the interviewees for approval. It was also sent to my university supervisor for feedbacks and after that the final version of the report was created.

4. Results

The focus of this study is on inner sourcing software products and we investigated the challenges, possibilities and potentials for inner sourcing at the company. The development work is carried out in different locations including Bangalore-India, Wroclaw-Poland, and Gothenburg-Sweden. Interviews were used to collect data in this case study and the interviewees were selected from the above-mentioned sites. Main development department teams which develop and operate Java, .Net, JavaScript, mobile applications and telematics platform solutions were interviewed.

The results of the analysis were then labelled and categorized for better readability. They were also sent to participants for reviewing. Later they were compared to the literature.

4.1. Challenges

In this section we answer the RQ1 which is “What are the obstacles and challenges of adopting Inner Source?”

The first part of our interview focused on the challenges and obstacles of Inner Sourcing in the company. Major development teams including Java, .Net, JavaScript, mobile apps and platform delivery were investigated.

Main problem according to all interviewed teams is the lack of contributions. Departments which tried Inner Source barely got any contributions and if there was any, it was in the form of bug reports or suggestions.

In this section, the challenges are investigated and categorized in five groups: organizational, people, product, infrastructure and process challenges.

4.1.1. Organizational Challenges

- **Managing and coordinating people**

One of the main challenges that most teams referred to, was coordination with people from other teams. In traditional projects, managers assign tasks to their team members but in Inner sourcing participants might belong to other department which the manager has no control over. This could affect the project speed and performance as sometimes there is a need for making quick changes but developers from other teams are not available.

- **Code ownership**

Today all software developed in-house is the property of the company unless specified otherwise in the contract with customers. Code ownership implies which team owns the code and has the authority to access, modify and distribute the source code. Interviewees think that code ownership becomes complex when multiple teams collaborate on a project. The code owner needs to be specified and it could be an issue if multiple teams had the same amount of contribution.

- **Maintenance responsibilities**

In open source projects many developers voluntarily contribute to the project. The contribution might vary from a single commit to the code repository to thousand lines of code. The person might leave the project and take no further responsibility. All interviewees agree that pure volunteer work does not work for critical projects in a company. There needs to be defined responsibilities both for development and operation of the services.

As can be seen in Figure 5, 82% of the current projects in the company are in maintenance phase. This could vary from small bug fixes to pure maintenance. The maintenance responsibility should be defined clearly for business-critical projects even if the code is shared. Furthermore, another big challenge is to define who should pay for the maintenance costs.

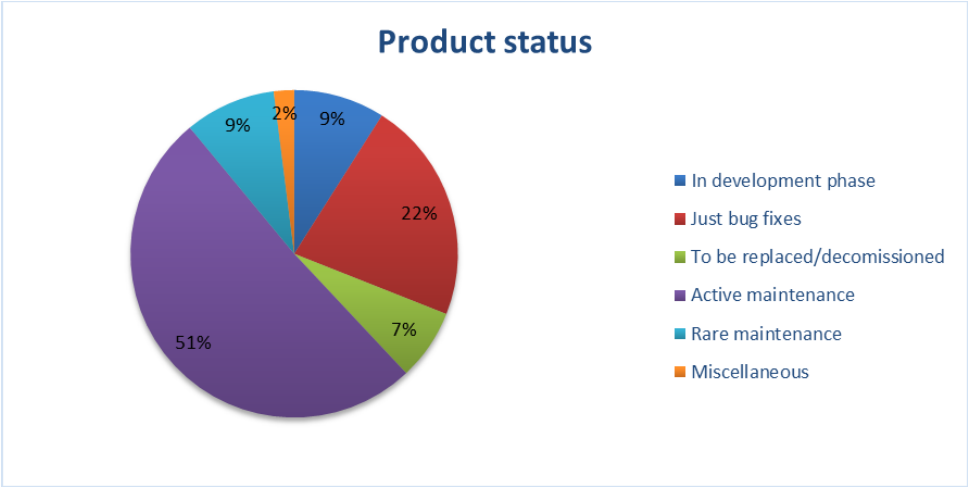


Figure 5-Current products status

- **Support from managers**

Our interviewees think Inner Sourcing must be supported by managers and without managers support it is not possible. Top and middle managers must be convinced that this is beneficial for the company and brings value. If they are not convinced, the resources and infrastructure for implementing Inner Source will not be provided.

- **Constant reorganizations**

Some interviewees mentioned constant reorganizations that occurred in the company during the past couple of years as a huge obstacle. Many people who were contributors or sometimes initiator of a project left the company and the Inner Source project left incomplete.

4.1.2. People and Individuals' challenges

- **Lack of motivation for contribution**

One of the teams had experience with inner sourcing and the main challenge they faced was the lack of contribution from developers of other teams. This was also mentioned by other interviewees as the main challenge. The reasons mentioned for lack of motivation are as following:

- Daily work pressure
 - Majority of interviewees believe that the main reason behind lack of motivation of developers for contribution is that they are busy running their assigned projects and they find no free time to dedicate to other projects.
- People mindset
 - Some people are not willing to share the code they developed and prefer to keep it to themselves.
- Finding no benefit in contributions
 - Some interviewees think they don't gain any benefit by putting extra effort to other projects.

- **Demotivated by others**
Interviewees mentioned that they lost their motivation after being left alone in the project and didn't receive any contributions.

- **Lack of domain knowledge**

Some areas in the company require specific domain knowledge which other teams may lack. This is a big obstacle for developers who are willing to participate.

4.1.3. Product challenges

- **Existing products architecture**

There are many applications at the company that don't follow the architecture guidelines and are not built for being shared or distributed. Some have a huge code base and modifying them is cumbersome as making a change in one section of the code may require change in many other parts. This could be a big challenge for inner sourcing as the product needs to be open to constant changes.

- **Risk of having poor quality code**

If code is not monitored carefully, there is a risk of having code with poor quality or inconsistent with different styles due to the fact that the code is written by multiple developers and each developer used his/her own style of coding.

- **Lack of documentation**

Many of the products at the company are not well documented. This is a big obstacle for Inner Source as developers may find it hard to only rely on code and have no documentation on the product.

- **Risk of formation of fork versions**

The changes to be made needs to be investigated carefully and the products that get affected by the change should be identified. This could be a challenge if the source code is open and people used it in other projects and different forks are formed which is derived from the main branch.

This could cause issues later for example in one case a team needed to modify the shared code and make some changes to it, but the changes could not be applied to the main branch due to the effect on other products. Over time the main branch changed and there have been several bug fixes while the fork branch remained unchanged and still has many bugs. After few years they faced a lot of problems and couldn't use the fork code anymore.

4.1.4. Infrastructure challenges

- **Different tools used by different teams**

Although the company has defined the assigned tools and software to be used but still old versions of the assigned tools are used by different teams across the company. Sometimes due to their demands, completely different tools are used. This could be an issue as inner sourcing requires different groups to use the same set of tools or compatible ones.

- **Security issues**

Some interviewees referred to maintain the security of the company data in Inner Source as a big challenge. In most cases access to specific data is limited to specific teams and projects have to ensure

this data is secured. Having the source code open threatens the security and solutions needs to be developed to ensure security of data is maintained. Furthermore, an interviewee shared his concern as if for example an employee from another organization has access to the source code but is not in any way involved or responsible in the project finds a security bug in the code and misuses this information against the company; it could have very bad consequences.

- **Access issues**

Some interviewees concern that technical issues might be encountered due to being on different network domains and it might not be possible to work on the same project due to networks and resources not being accessible to everyone.

4.1.5. Process challenges

- **Project planning**

One of the challenges of inner sourcing for project managers is that they need to plan the time and budget ahead. Planning is done based on the project size, team structure and number of people involved and their skills. However, this is totally different in Inner Source where individuals from all over the company could join the development team at any time and contribute. Cost estimation is very difficult and cost sharing with other teams is more complex. Currently, it is not possible for everyone to use any WBS (Work Breakdown Structure) to report their work. Modifying the system to support Inner Source could be a challenge.

- **Nature of commercial projects**

Many interviewees refer to the nature of business projects as the main obstacle. In Open Source, the focus is not on the commercial aspect of the product but in a business environment, the financial value of the product plays an important role. Therefore, there is constant pressure to meet the budget goals and scheduling requirements. Deadlines are important for business and customers can't wait for volunteer resources to be provided from Inner Source community to provide a feature or fix a bug.

- **Communication and coordination**

Many interviewees find the need for communication and coordination with other participants from other departments quite challenging. As the team scale grows coordination could become overkill since all changes need to be discussed and agreed upon publicly. People find it cumbersome to coordinate with others when they only need to commit a few lines of code. Communication might become slower as sometimes there is a need for having meeting and live discussions and it is difficult to find available slots for meeting with people from other teams especially if they are located in different geographical locations. Furthermore, managing different and sometimes conflicting ideas become harder as the number of people involved increases.

- **Lack of visibility of the existing projects**

One reason that many referred to was that they are not aware of the current projects inside the company. Lack of information and transparency of the ongoing projects is a challenge for developers. These projects must be announced and visible to other teams inside the company who would like to join and contribute.

4.2. Product

In this section, we focus on RQ2 which is: “Which software components/products are best candidate for inner-sourcing?”

The second part of the interview is focused on finding criteria which are important to interviewees in identifying suitable products for inner sourcing.

4.2.1. Usability

The company we studied owns several applications. According to the statistics, 87% of the software is developed internally as displayed in Figure 6. In such environment, reuse possibility is very high. Many components can be shared among different applications.

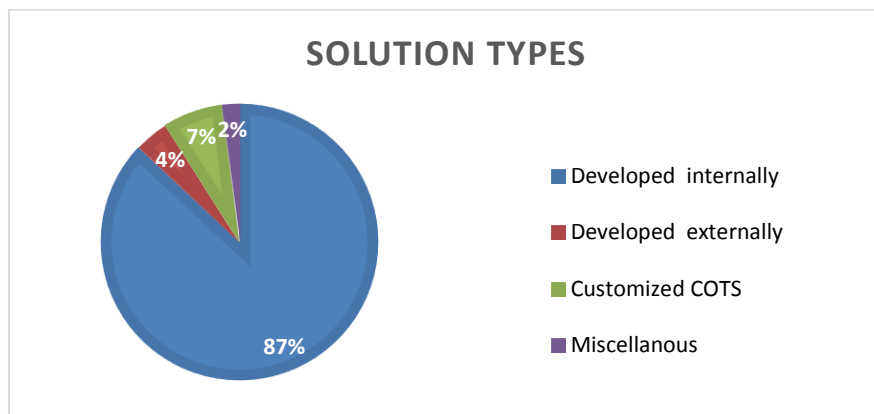


Figure 6-Company software

- Usage:
All interviewees agree that usability of the product is the most important factor to consider as high usability of the product motivates users to participate and improve the product. Products with a larger variety of stakeholders have a higher chance of attracting contributors. However, products that are specific to a domain or unit have less chance of attracting others. Furthermore, according to interviewees there have been several cases when the code owner team didn't have time or resources to fix the program for users while other users had knowledge and resources to fix the problem but due to code ownership issues they could not modify the code.
- Re-use:
Moreover, according to interviewees reuse possibility is very important. Several duplicate programs and functions are created in different organizations over years but due to not being visible other teams are constantly reinventing the wheel. This is both time consuming and costly for the company. Inner sourcing the products with high reuse possibility helps resolving this problem.

Usability & Re-use possibility	Not reusable
100%	0%

4.2.2. Nature of the product

- 87% of the interviewees think that a good candidate product must be “company specific” and a unique product that does not exist outside the company. They think that since the company main business is producing vehicles and not IT solutions and its business is to serve the needs of the parent company, the focus must be on using the products available in the open source or on the market and are cost efficient to buy instead of developing inside the company. One team had an experience of developing DLF UI components, but the project became so big that it needed to be handled by a separate company and they found out that there are similar solutions on the market which are more cost effective to buy rather than developing internally.
- 12.5% of the interviewees believe that products which are not company specific and are aimed at more general purposes are good candidates as long as they have value compared to what already exists on the market or if such product does not exist outside on the market, it might be logical to develop it in-house. Such products can even straighten the path towards open source.

Company specific	General use software
87.5%	12.5%

4.2.3. Criticality

- All interviewees think that a business-critical product can be Inner Sourced as long as there is a core team dedicated to it and have a formal maintenance setup. For such products, pure volunteer work does not work. They don’t see any problem in inner sourcing a critical product only if it is operated and supported by a dedicated team as the availability of resources is vital for such products.
- For other non-critical products such as supplementary tools, inner sourcing is possible without the need to have a dedicated support team.

Note: Security measures should be considered when inner sourcing products specially the critical ones.

Note: Security measures must be taken to protect the data when the code is shared among different departments

4.2.4. Architecture

All the interviewees agreed that a modular architecture is the best solution for a product that is going to be Inner Sourced. Modular architecture makes the product easy to modify. Each module can be tested separately, and it is more readable than a big chunk of code. Beginners who want to participate in product development don’t need to study the whole code and they can have a faster startup by focusing on specific modules instead of the whole program as the program becomes easier to understand. Furthermore, the program divided in different modules enables developers to work on modules independently. Product architecture is of great importance when it comes to Inner Source candidate selection among the existing products. If the plan is to Inner Source an existing product, change the existing architecture might be far-fetched.

One of the recently introduced architectural patterns called “Microservice Architecture” was proposed by interviewees as the preferred architecture style for Inner Source products which implies designing software as suites of independently deployable services [57]. According to Fowler, “The Microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery.” [58]

Oppositely, in monolithic software different parts are bind together and a change made to a small part, impacts the whole application and requires the entire application to be rebuilt and deployed. Maintenance of monoliths is quite hard and over time it’s not possible to make changes to a single part without impacting other parts in solution. The same goes for scaling as it requires the entire application which implies the need for more resources [57].

Using micro services, it is possible to deploy the software more frequently. Interviewees think using micro services, the program does not have to use one technology stack and a service can be developed using another programming language [57]. This is a big advantage for Inner Source as it facilitates involvement of programmers with different skills. On the contrary, as the monolithic code base gets larger and larger, it becomes almost impossible to change the technology stack of the whole program. Using micro services, the changes could be made gradually and independently without affecting other modules.

According to interviewees, a monolithic program is hard to understand and change. Furthermore, the whole program must be redeployed after each change whilst using micro services each service can be build and deployed independently. Program is easier to maintain as a fix in one service doesn’t affect other services. In addition, it enables developers to be able deploy frequently which improves delivery time.

Loosely coupled components can serve the purpose best as they are less dependent on other components. They could integrate with other components. Their maintainability increases as they can be replaced by other implementations. In addition, services must be security persistent.

Components which have clear input and well defined have a clear interface for further development. VGT common platform API is made using microservice architecture which makes it a perfect candidate for inner sourcing.

The architecture of shared product must support collaborative working. Microservices architecture seem to be a good fit for Inner Source as each team or individual can work on a single service and each service has only one single responsibility which makes it easy to modify and maintain. The services are loosely coupled so they can be built and deployed independently. Furthermore, each service can be developed using a different technology depending on the skills available [57]. In addition, modularity increases the reuse possibility which is one of the main goals at the company. According to the interviewees the experience from previous Inner Source projects indicated that it was a successful design and worked very well for small self-organizing development teams.

4.2.5. Maturity

93.7% of the interviewees think it is better if the product is in its initial phases of development for example close to version 1.0. The reason for this is that it is easier to understand the program in earlier stages before it gets too complicated. In this stage, the main requirements and features are collected but there is still room for adding new requirements. As the project progress it becomes more complicated and difficult to understand especially for new comers.

Interviewees argue that if it is just a pure idea, it is still vague and announcing it in such a huge company is not possible but if it is in its initial phases for example close to version 1.0 is ideal specially if there is a demo or prototype for it, it is easier to introduce to others and attract developers and stakeholders.

However, 6.25% think that a mature product which is stable and well documented has success chance since many are already familiar with it. Therefore, the chance of contribution for improvement and bug fixes is higher. The precondition is that it should have the possibility to grow, improve or extend.

Not started	Initial phases	Mature
0%	93.75%	6.25%

4.2.6. Technology

81.25% of the interviewees think that the technology being used by an Inner Source must be one that is used commonly inside the company by developers because if it is a technology that is only used by few developers then the Inner Source project may not get the attention it requires and fail. Some even think that it should be popular outside the company too as it gives developers the opportunity to learn from developers outside the company and also be able to use the knowledge they gained from the project later on in their future careers. Java, .Net and JavaScript were named as technologies which are commonly practiced inside the company and outside as well.

One of the interviewees mentioned that they have more than hundred java applications in their department while there are only three COBOL applications which denotes the popularity of Java in the department and decreases the probability of facing “No contribution” issue which many have referred to as the main challenge for Inner Source.

18.75% believe that it doesn’t make any difference if the technology is widely used or not. They believe that even though some technologies might not be used as common as others but still some critical applications are made based on them. Mainframe is a good example. Although it is not based on a recent technology and the number of experts is not many, but it is still highly used at the company and could attract experts.

Commonly practiced Technologies	Other technologies
81.25%	18.75%

4.2.7. Size

Size of the product is determined by many factors and everyone might have a different interpretation of what is a big project. No of Lines of code by itself does not determine size of a product. Moreover, a program with 1000 lines of code could be much more complex than a product with 10000 lines of

code. Interviewees suggested considering all the following factors to determine the size of the candidate product for Inner Sourcing.

- No of requirements and features
- Code complexity
- No of lines of code

All interviewees agree that as long as the product has enough prospective features and functionalities and has the possibility to grow, then the size is not an issue. Obviously, it should not be too small because then there is no point in inner sourcing and seeking assistance for development. Moreover, if the product is created based on modular architecture, then the size of the code should not matter because each developer can pick up a module and work on it.

4.2.8. Other criteria for product

Since inner sourcing might require extra efforts and resources, not all products worth it so other criteria including costs and overheads must be considered before deciding to go for Inner Source. Participants mentioned other criteria for the Inner Source product including:

- Well-documented
- Polished code
- Fun to work with

According to the above criteria some of the company products where suggested as good candidates for inner sourcing during workshops with teams.


4.3. Adopting Inner Source by the company

RQ3 is addressed in this section which is: “How can the company adopt Inner Source?”.

Inner sourcing is a new approach for the company and needs to be adapted to the needs of the company. In this section, the interviewees were questioned regarding the implications of inner sourcing for the company. The pre-conditions, processes and infrastructure that need to be established for Inner Source and the challenges they address were discussed. The focus was on the main processes and tools that are more specific to Inner Source. Please note wherever tool names are mentioned, it is not the tool itself that is interesting but rather its functionality.

4.3.1. Tools and infrastructure


- **Common set of tools**

Challenges to be addressed	
	Different tools used by different departments

One of the challenges interviewees mentioned was the use of different tools or versions of the same tool in different departments. This could create chaos by having to deal with incompatible tools.

Therefore, a common set of tools needs to be agreed upon to be used by everyone who is going to participate in the Inner Source project. Each project might require different tools depending on the needs. One way to achieve this is to share the tools on software forge. Tools could vary among projects but within the same project the same tools must be used to avoid incompatibility issues.


- **Software Forge**

Challenges to be addressed	
	Developers not aware of the ongoing projects (Lack of visibility of the existing software products)

Interviewees mentioned that they are not aware of the ongoing projects in the company and the reusable assets developed in other departments are not visible to them. Due to this problem lots of re-work is being done at the company which imposes extra costs. The solution to this issue is to use a web-based dashboard where everyone in the company can see the ongoing or finished projects with information about them. A central search function is required to index the products. The dashboard could be used to publish news and other information and host discussion forums and other shared contents. Software forge provides services to facilitate collaborative software development. The services include project portfolio such as bug tracking system, version control system, mailing list and wikis [20] [41]. Currently such platform is missing.

Visualization of the information helps better understanding the status. Stakeholders need to know about the product status as well as developers do so the outcome and roadmap could be shared with stakeholders. If possible a demo of the product could be made to visualize the product. The progress of the project could be shared by stakeholders, so they can get an overview of the progress which makes real-time planning easier.

- **Version control system with code review and QA capabilities**

Challenges to be addressed	
	Risk of having code with poor quality

One of the most important tools required is a common repository. Developers need to have a common work space to share and sync the code. Such a tool exists at the company but for Inner Source it is important that the tool has QA (Quality Assurance) capability that automatically reviews the code. This is required to block the bad commits to the master branch. Needless to say, that security over code distribution and usage is of high importance. Only authorized users may commit to the common repository. This is the area that Inner Source differs from OSS and needs to be tailored to fit the commercial context.

A good example of a tool with QA capability is Gerrit ¹ which is an extension to GitHub repository. Submitted code goes to a pending location until it is reviewed and approved before it is finally checked in to the master repository. Figure 7 shows how Gerrit works.

¹ Gerrit code review: <https://www.gerritcodereview.com/>

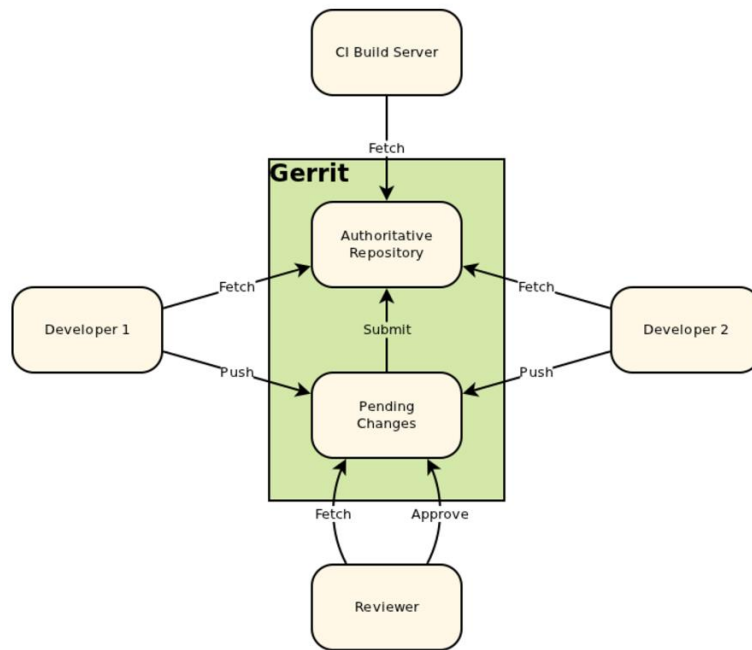




Figure 7 - How Gerrit works [47]

- **Automation Tools**

Interviewees agree that automated tools for code review, testing and builds are crucial. It removes the urge to perform the repeated tedious tasks manually and prevents human mistakes.

In addition, there are many other tools to benefit from for communication purposes, information sharing, archiving and storage, code review, bug tracking, build automation, etc.



- **Accessible work space**

Challenges to be addressed	
	Access issues
	Security of data

Most participants think working and accessing resources on the company network is not optimal for shared projects due to firewalls and access limits. Lots of coordination with other teams such as firewall, database, web sockets, security team and etc. is required if multiple teams need to share resources which can discourage developers.

It must be ensured that the infrastructure such as computer network proxies or policies do not prevent developers from working on a common work space. If developers find working on other projects too cumbersome, they will easily get demotivated. To avoid this issue, using Cloud services such as Amazon or Azure cloud was proposed as an alternative to host the applications. Obviously, still access controls must be in place and developers may only access specific development environments. In addition, security measures should be taken to avoid leakage of sensitive data as a result of visibility.

- **Documentation**

Challenges to be addressed	
	Lack of documentation on products which makes joining other developers hard
	Lack of domain knowledge

- **Product documentation**

Lack of documentation or not enough documentation on products hinders participation of enthusiastic developers and newcomers. Interviewees believe documents make it easier to follow up on a project if one joins the team later in the development phase. Therefore, there is a need for having documentations which everyone can see and contribute to. It could be shared on the forge or anywhere on the intranet. A wiki was suggested and also the electronic discussions/decisions related to products could be archived in a forum to serve as product documentation for further reference.

- **“How-to” documentation**

Since Inner Source is a new way working for employees, participants believe there is a need for guidelines which helps newcomers to understand how to work in an Inner Source project.

4.3.2. Adoption model

First, the two adoption models, Project-based and Infrastructure-based explained in section 2.5.1 were discussed. Interviewees claimed that the adoption model selection depends on the product. Critical products require dedicated teams to support the customers and manage the asset considering the big picture. In this case, providing support is mandatory. The critical software needs to have a clear owner that is responsible for maintenance of the product and takes care of the other related tasks around it such as planning, documentation, test, release, etc. and shares it to all other stakeholders. The reuse possibility and the needs of other user groups must always be considered in design decisions

On the other hand, there are many other software products that are not business-critical, but they are required by many projects and sharing them prevents redundant work and re-inventing the wheel. Interviewees think that sharing software facilitates the reuse. For these utility tools and common libraries, support is optional and management overhead is kept at minimal level. Individual developers can share their code using the provided infrastructure by the company to host their project and let other projects benefit from it.

Therefore, the adoption model must be a mixture of both infrastructure and project-based models and the organizations selects the appropriate one depending on the criticality of their products.

Defining the right accesses for users and developers is very important. Project-based components have limited access while Infrastructure-based projects are open to other users/developers.

Note: Code ownership must always be considered and discussed as sharing/reusing the code which is owned by customer might not be permitted.

4.3.3. Processes

4.3.3.1. Development processes

Open source requires an agile and collaborative way of working which is referred to as Bazaar style by researchers [11]. In Open source there is no single development process to follow and each open source project has its unique process and way of working [16]. Interviewees claim that Inner Source follows the same approach as OS and those heavily controlled traditional development processes do not work here. However, different organizations within the company need to align their processes to be able to cooperate efficiently. In addition, for a company to be able to satisfy the customers and meet the deadlines, different organizations within the company must follow a mutual working process.

Today Scrum which is an agile methodology is dominantly used by development teams in the company. It guides their way of working and task division among team members. Scrum roles consist of product owner, scrum master and development teams that are maximum 8 people and are self-organizing meaning that each team plans their own backlog. It works well for concentrated teams. However, in Inner Source team members might be distributed in different geographical locations and it is not possible to follow all scrum practices such as daily stand up meetings due to time zone differences or simply because Inner Source developers belong to other organizations and have other meetings/tasks to attend to. They believe they can still benefit from many of the Scrum practices such as backlog concept and sprint reviews while they might need to adjust some of them to suit their Inner Source project. In addition, since face to face meetings and daily standups are eliminated, the need for having transparent communication is obvious. Therefore, documentation is highly required.

Self-organizing in Inner Source is not the same as open source and needs to be tailored to company context since projects are tied to deadlines and budget constraints. Therefore, it is fair to say developers should organize their backlog within the project deadline and budget. The choice to keep the roles must be made within the team.

- **Release management**

Interviewees mentioned the Linus Torvalds style of development. He created Linux kernel and one of his famous quotes was “Release early, release often”. This approach was also taken during UNIX development [11]. It allows users and prefer developers to be able to provide feedback earlier in the lifecycle as opposed to the traditional approach were users could provide their feedback earliest during beta releases. This could prevent a lot of bugs in early stages and save a lot of time and cost. This approach is divided to three phases:

- **Continuous integration**

According to Fowler, “*Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly.*” [60] This way the code all developers use is always synced with the main codebase. This implies using a version control system and verifying the changes by performing automated tests constantly.

➤ Continuous delivery


According to Fowler, “Continuous Delivery is the natural extension of Continuous Integration: an approach in which teams ensure that every change to the system is releasable, and that we can release any version at the push of a button.” [60] The term was first used in agile manifest and is one of the agile main goals [61]. It involves delivering code to an environment close to user’s which could be QA or Prod. However, deploy to production environment is not done automatically and it’s up to team to decide whether to do it or not [37].

➤ Continuous deployment

Continuous deployment implies deploying software more frequently which means it is available to users as soon as the code is developed. It is an extension to continuous delivery as the code will be deployed to production automatically after it passes the tests [26]. It has many advantages and the most important one is that users can provide their feedback continuously while the program is being developed which prevents bugs being detected in the very last minute. In addition, the less time there is between deploys it is easier to find what has been changed since the last successful build. Time to deliver a feature to market decreases and the business risks minimize due to having shorter time to market. The code is built, tested and deployed automatically after each change.

There are many automated build tools to benefit from. One of them is Jenkins ², which is an open source tool that is used to automate all sorts of tasks such as building, testing, and deploying software. It runs the tests and if all tests are passed, the code will be checked in.

• **Maintenance**

Challenges to be addressed	
	Maintenance responsibility

➤ DevOps

Developers follow a project model with the aim of delivering a software. After completion the software is handed over to maintenance team and the development team job finishes there. Microservice proponents tend to avoid this model, suggesting that a team should own a product over its lifetime which implies the development team takes full responsibility for the software in production. This brings developers into daily contact with users and they can see how their software behaves in real time production environment and they must take over at least some of the support responsibility [57]. DevOps goal is to close the gap between Development and IT Operations to enhance the deliveries. It is a set of practices and establishing a culture of working together in the organization.



In short DevOps implies that the one in charge of development must make sure that the software works in production environment as well. Currently in the company the responsibility for delivery including development and operation is carried by separate teams but if development team can participate in operation it would be more suitable for them since they have a better understanding of the program and they can change it based on the needs and ensure if it still works. According to interviewees this works best in Inner Source as if a team needs to change the code for their own demand, then operation cannot take the responsibility

² Jenkins continuous delivery <https://jenkins.io/>

of it but if it the team itself is responsible for the operation of their code then there is no problem.

4.3.3.2. Governance processes

- **Quality control**

Challenges to be addressed	
	Poor code quality
	Formation of diverging code forks

Quality control is an important step in any software project. Interviewees agree that the code must be polished before submission as it must be readable for others who are going to use or modify the code later. The following approaches were suggested for maintaining code quality:

- **Code standards**
Coding standards needs to be established and followed by developers. This helps to maintain the code consistency and increase the code quality.
- **Code review**
One of the activities to enhance code qualities is code review. Participants suggest having a team responsible for control and reviewing the code. Code review is very common in open source community and industry [59]. It can be done using many approaches including manual approached such as pair programming or automated ones using code review tools. Automated tools enable the possibility for asynchronous working. There are many automated tools to benefit from and one is SonarQube³, which is the open source platform to inspect code quality of the programs. It enables developers to detect bugs and vulnerabilities as well as to decrease code smells, in more than 20 different languages. Statistic code analysis is also considered as a part of code review and its purpose is to find the security vulnerabilities early in the code. This feature is also embedded in the tool. Furthermore, having metrics on the code helps other team members to get a better overview of the status of code.
- **Bug tracking**
There is a need for a bug tracking mechanism so that developers, testers and users can register defects and later track the status of the defects. Using this system, the bug can be assigned to a developer to fix. Bug tracking is sometimes integrated into test management tools, but it could also be a separate system. Currently Jira ⁴ is the assigned bug tracking tool at the company
- **Automated tests**
Testing is an important activity in software development. Automated tests can be extremely helpful in an Inner Source project to replace the repetitive and time-consuming manual tests. Interviewees suggest having as many automated tests as possible as they are more efficient especially in scenarios where code changes frequently. These tests should include backward compatibility test as well as integration tests to validate that components interfaces work as


³ <https://github.com/integrations/sonarqube>

⁴ <https://www.atlassian.com/software/jira>

before. In addition, automated tests are a precondition for continuous deployment which we will discuss in the next section.



Using Test-driven development (TDD) approach is also recommended which implies having tests in place before coding.

- **Change Management**

Challenges to be addressed	
	Formation of diverging code forks

Changes can lead to improvements in the software if handled properly but they might also create inconsistency or instability issues in the associated components or products. Therefore, it is vital to have an efficient change management processes in place covering both pre and post change procedures such as request, review and publish change procedures. All changes must be coordinated and controlled with related teams. The process should address the roles. This is especially of great importance in Inner Source projects due to its nature. Furthermore, formation of independent code forks must be prevented as much as possible.

- **Project management**

Challenges to be addressed	
	Project planning issues
	Managing and coordinating people

Interviewees agree that traditional project management approaches do not work in today IT world where requirements are constantly changing, and we need to seek agile ways of project management. Project management in agile is different from the traditional model as not all the tasks and milestones are planned upfront. However, there is a need for project management in every project even in the open source projects but there is no unified method or practice that works for all projects rather it depends on the organization setup and processes. Cost management methods needs to be specified and time and budget allocation strategies must be established to support the distributed development. According to interviewees the following project management approaches must be considered in Inner Source:

- **Project planning**

- ✓ **Activity planning:**

ISS products have a long-term vision and it must be ensured that features do not deviate from the initial vision. It is mainly the responsibility of the core team to consider the long-term plan. All activities and backlogs must be created considering the overall direction. When other departments request new features and components, the core team must review the request and approve the feature having the vision in mind.

Furthermore, it is important that individuals' projects and activities are prioritized so they don't interfere with each other.




- ✓ Resource allocation:
In Inner Source resources will not be assigned to tasks but rather they select tasks themselves. Implementation of the requirements requested by the other units, can be done mutually by business units in collaboration with the core team.

➤ Project control

- ✓ Cost control:
Currently cost management system does not support registering contributors from all departments which needs to be changed since there is a need for a WBS (Work Breakdown Structure) that could be used commonly by all the people who work on the same project.

4.3.4. Organization and communication


- **Communication and Information sharing**

Challenges to be addressed	
	Communication and coordination issues
	Lack of documentation on products which makes joining other developers hard
	Lack of domain knowledge

One of the most important factors in the success of Inner Source project is contribution of the developers from all around the company. It is vital that the project is visible to as many people as possible and the latest information and updates are shared. A Wiki page or a dashboard for the project to display the project information such as latest status, updates, bug fixes, people involved, point of contacts, etc. is very useful. Currently, Confluence⁵ is used which is a team collaboration software used for keeping the team work in one place. It has many features including notifying users when a document change.

Use of internal forums and channels is recommended as discussion can take place virtually and it replaces face to face meetings and daily discussions to some extent. People can read on the topics shared in the forum which they are interested in whenever they have time. Besides, communications are stored electronically for further reference which is a positive point. Currently Slack⁶ is used by VGT organization. It is a developer centric tool and using the tool it is possible to open a new channel and invite people. Besides, one can subscribe to channels of interest. Mail lists could also be used as a way of asynchronous communication. Participants think that when collaboration is distributed, there is a need of overhearing what is happening, and others can see what is going on.

- **Responsibility Sharing**

Challenges to be addressed	
	Managing and coordinating people

⁵ Confluence software, <https://www.atlassian.com/software/confluence>

⁶ Slacks collaboration tool https://slack.com/intl/en-se/?eu_nc=1

Currently agile and specifically Scrum is the methodology used by development teams at the company which guides their way of working and task division among team members. The roles are Scrum roles consisting of product owner, scrum master and development teams that are maximum 8 people and are self-organizing meaning that each team plans their own backlog. Self-organizing in Inner Source is not the same as open source and needs to be tailored to company context since projects are tied to deadlines and budget constraints. Therefore, it is fair to say developers organize their backlog within the project deadline and budget.

In Telematics organization, DevOps is used which implies that the team who developed a component or service is also responsible for its operation in production. If this approach must be applied in Inner Source, then the developers are responsible for operation which is not exactly in line with volunteer contribution idea initiated by Open Source concept where tasks are selected voluntarily by developers. Furthermore, maintenance responsibility needs to be defined in advance or it is not possible to accept high Service Level Agreements (SLA)/Operation Level agreement (OLA). It needs to be adjusted to match the capabilities of the team handling the project. The level of supports provided depends on the selected Inner Source model whether it is Infrastructure-based or project-based. In addition, clarification of role expectations upfront results in better performance outcome.

In OSS decision-makers are not defined based on their hierarchical position in the organization but rather based on their merits and contributions to the community. This approach needs to be considered in Inner Source as well. The degree of acceptance depends on the current culture of the organization. Some teams seem to be more open while others are not, but the interviewees think that this shift does not happen over a night and takes time to adapt it. A more democratic way of decision making must be chosen for example one participant mentioned using voting for new features. Interviewees believe that volunteer task pickup might not be fully applicable to a business environment with tight deadlines and customers. In open source the number of developers is usually large, but it is not the case in a company where resources are limited. However, volunteer task selection could be applied to a degree as it is done in Scrum today using a common backlog and more coordination.

➤ Roles

Interviewees suggest having the following roles in the Inner Source environment in addition to the current ones. Other roles may appear after more experience with inner sourcing is gained depending on the needs.

- Product owner and governance organization

Interviewees suggest having a governance committee that monitors the project and directs the project towards the planned goal. The product owner could be the individual or team in charge of the product who defines the goals and the backlog accordingly.

- Coordinators

In a distributed environment, coordination between teams is crucial. The coordinator must align the work between the core team and stakeholders.



- Mentors

Mentorship is required to teach others about the product. Interviewees believe that if newcomers find the product too complicated and are not able to keep up with the pace there is a good chance that they get demotivated and leave the project. Mentoring could also help interested developers to address the problem “lack of domain knowledge” as one of the challenges mentioned by interviewees.

- **Culture**


Our interviewees agree that Inner Source requires a cultural shift in the organization. It requires transparency as well as willingness to share knowledge. This is different in the traditional way of working where only team members are aware of the code and status of the project. In Inner Source, similar to open source communities all necessary information is available to other participants. Openness to change and critics is of great importance since during peer reviews the code might be subject to many changes.

- **Management support**

Challenges to be addressed	
	Lack of support from management
	Lack of motivation for contribution

Obviously starting an Inner Source project or engaging in projects in other departments requires management consent. If developers have no free time to join the project or their managers do not agree with working on another department project, then Inner Source will fail. Project budget and resources needs to be approved by top managers. In addition, managers play a great role in promoting the culture and motivating people to join the project. The management board must convey this message that they appreciate this innovative way of collaboration and support it.

- **Motivation**

Challenges to be addressed	
	Lack of motivation for contribution

Many interviewees mentioned the lack of motivation for contribution as the main challenge. Some people say that they get motivated when their contribution is valued, and their implemented ideas result in improving functions which they can also benefit from. Business units could be motivated to contribute to development if they find their suggestions are appreciated and their ideas are reflected in the output.

Some participants find working with other departments interesting and think it can help them learn from other colleagues and enhance their network of contacts. In addition, getting known could lead to better job opportunities in the company.

Basically, a mechanism for attracting professionals and required competences is required. A reward system based on individual's contribution was also proposed by some interviewees.

5. Discussion

In this section, the results of our findings which provides answers to the research questions are discussed.

5.1. Challenges

- RQ1: What are the obstacles and challenges of adopting Inner Source?

In this study we have identified several challenges associated with adopting Inner Source. They are classified into different categories. The main challenge mentioned by all teams was lack of contribution and we decided to investigate it more thoroughly to understand what the reasons behind this obstacle are.

The results of our analysis are demonstrated in the Figure 8 which is a cause and effect diagram known as fishbone diagram. The challenges were divided into five main categories: process, product, organization, people and technical challenges.

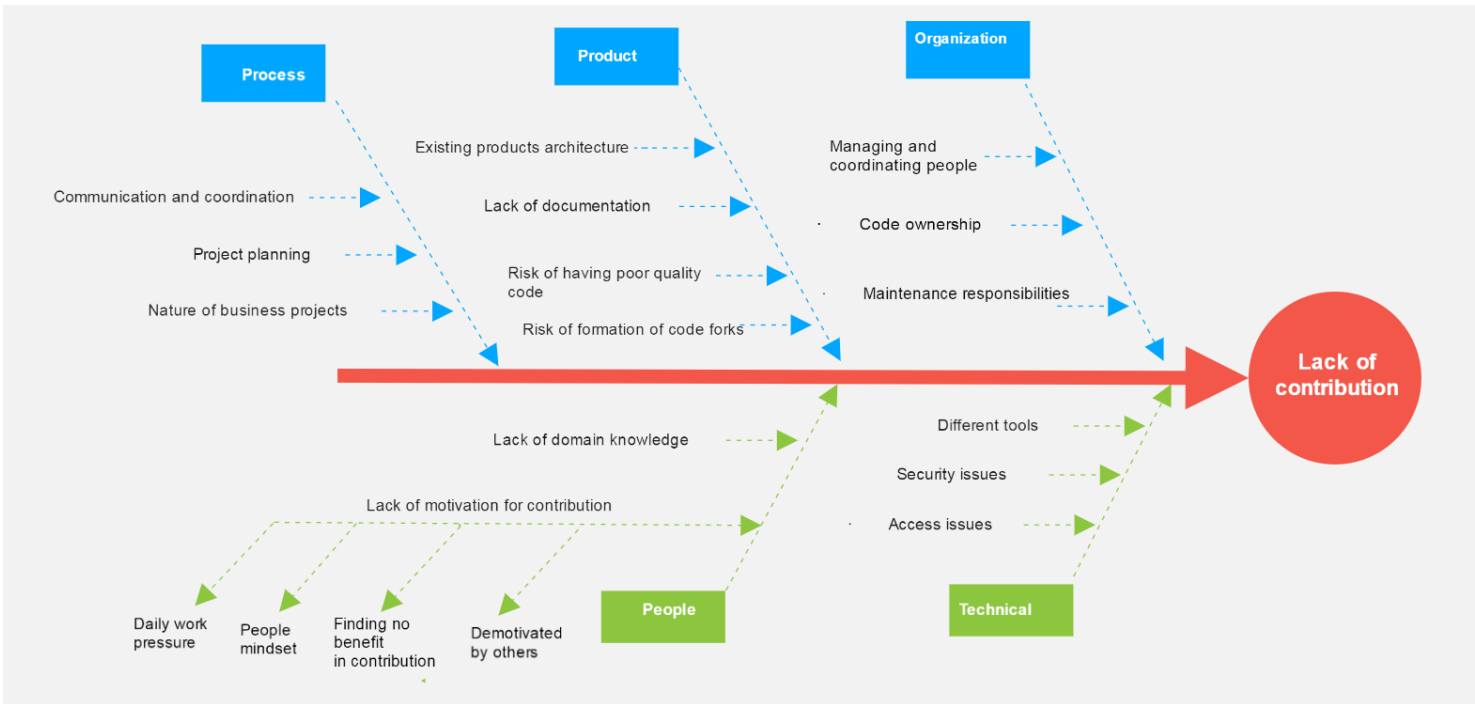


Figure 8- Summary of Inner Source Challenges at the company

Some of the challenges found in this study were also identified by studies mentioned in Section 2.3 which were conducted in other companies. The context of those studies is not discussed in detail in the reports. Various factors might impact the challenges people face in each case and each company is unique. Therefore, comparing the challenges without knowing the exact context is not right.

The company must consider the above-mentioned challenges and plan to remove the obstacles to increase the contribution of employees and facilitate the adoption process.

5.2. Product

A major step to be taken by the company in Inner Source adoption is identifying the suitable product. I interviewed professionals in the company who had experience with Inner Source or are willing to adopt Inner Source in order to find the criteria for product selection. Then I compared the outcome to the literature. However, there were only few studies available with the focus on this topic.

In our study, we found that products with high usability are the best candidates for inner sourcing. Gurbani et al. [49] also claims that the product must be needed by many stakeholders in order to attract contributions from users, teams and other dependent projects. Furthermore, they can benefit from other teams' knowledge and expertise. Having multiple stakeholders, means there is a lot of interests in the product and it is a good way of pooling the resources [17] [49].

In addition, most interviewees think it should be a "company specific" product rather than common tools or utilities which also justifies the in-house development of the product. Stol et al. found that the product type could vary widely as in Philips it was a large shared platform while in Rolls Royce it was a tool for designers [50].

Business criticality of the product was another criterion for Inner Source product selection and all interviewees agree that a business-critical product can be Inner Sourced if it is managed and supported by a core team under the condition that security measures are considered. Non-critical products such as tools may be Inner Sourced without those preconditions. Stol et al. reported the results of three case studies done in three large organizations including Philips Healthcare, Neopost and Rolls-Royce. They suggest that the start point should be a seed product that has a significant business value to the company [50].

Another criterion was architecture. According to interviewees modular architecture and use of loosely coupled components was the preferred architecture. Specifically, micro service architecture was suggested which is a new pattern that extends the benefits of modularity. This is in line with the results of a study by Stol et al. which suggest the product should have a modular architecture [17].

Maturity was the next criterion. All interviewees agree that there must be a base product available to maneuver on. When it is just in the requirement or "Idea" phase it is harder to attract developers than when an initial implementation is produced. This confirms the results of the study by Stol et al. which suggest that the product should be an existing one which is runnable. The reason is that starting an Inner Source project from scratch is quite difficult [50]. Raymond states that one cannot code from scratch in bazaar style. It is possible to debug, test and enhance the code but originating a project from scratch and expecting to attract developers is quite hard since developers need to have something to play with [11]. Wesselius claims that the seed product must have clear specifications so that it can be outsourced to other development teams [25].

On the other hand, Gurbani et al. hypothesis was that lacking a seed product, research or technology is a very good start point [49] [17]. Our study results do not confirm this hypothesis.

Majority of interviewees preferred products developed using common technologies because they believe the number of available experts in the area has direct impact on the level of contribution. On the other hand, three people argued that even technologies that are not as common but still used at the company can be a candidate for inner sourcing too.

According to interviewees, size of the product does not matter as long as it has a modular architecture and has the potential to grow. The results are summarized in Table 7.

Reusability	High	100%	Research Results: Gurbani et al. [49] :confirms
	Not reusable	0%	
Product type	“Company specific”	87.5%	Research Results: --
	General use software	12.5%	
Criticality	Business-critical: if support is guaranteed	100%	Research Results: Stol et al. [50] Product type varies a lot
	Non-critical: if inner sourcing adds value	100%	
Architecture	Modular	100%	Research Results: Confirms: Stol et al. [17]
	Non- modular	0%	
Maturity	Initial phases	93.75%	Confirms: Stol et. al. [50] , Wesselius [25] Not confirmed: Gurbani et al. [49] lack of seed product is a good start point
	Mature	6,25%	
Technology	Commonly used Technologies	81.25%	Research Results: --
	Other technologies	18.75%	
Size	Does not matter if architecture is modular and there is possibility to grow.		Research Results: --

Table 7- Product criteria according to our study vs. other studies

5.3. Adoption model

5.3.1. Project based vs. infrastructure based

According to the results of this study, selecting between project-based model and infrastructure-based model is dependent on the product. If a product is business critical, having a core team and support team is mandatory therefore, it is categorized under project-based model. On the other hand, products such as utilities, scripts and libraries which are not known as business-critical fall into infrastructure-based group where support is optional and there is no core team.

We identified a need to combine the two available models based on the product type to address different needs of the company according to Figure 9.

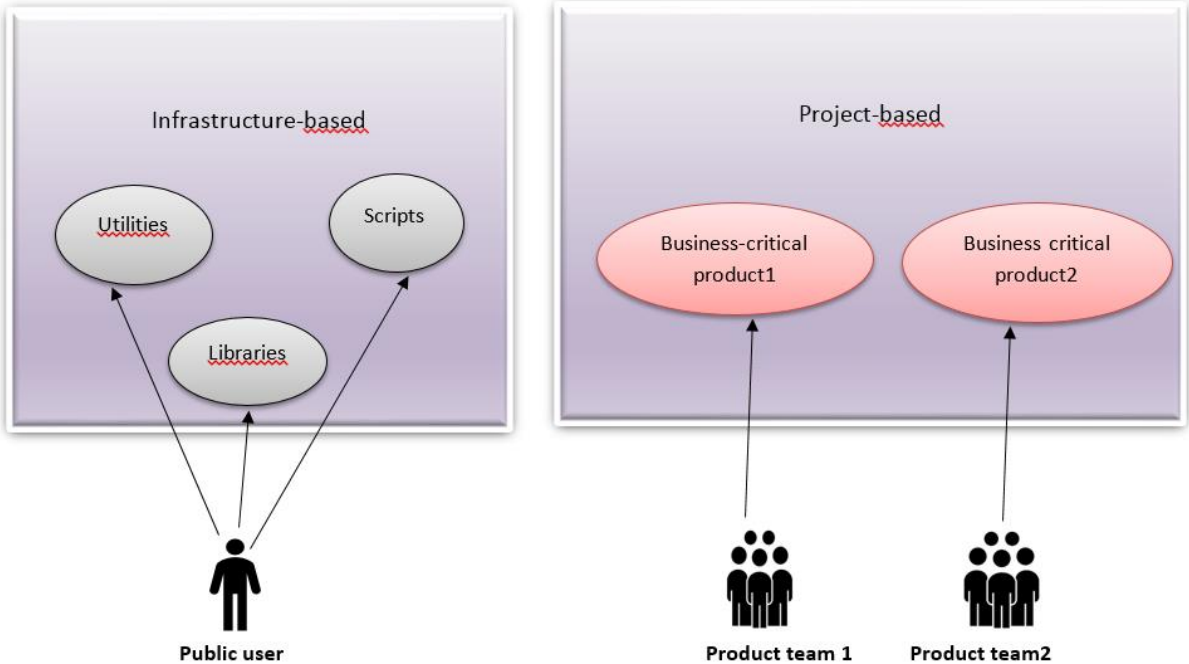


Figure 9- Adoption model at the company

5.3.2. Inner Source adoption by the company

According to the results of this study we summarized our findings on the adoption framework to four categories, Product Identification, Infrastructure and Tools, organization and people, development processes and governance processes. A set of practices and approaches were identified to implement the Inner Source within the borders of the company considering the challenges mentioned before and the company requirements.

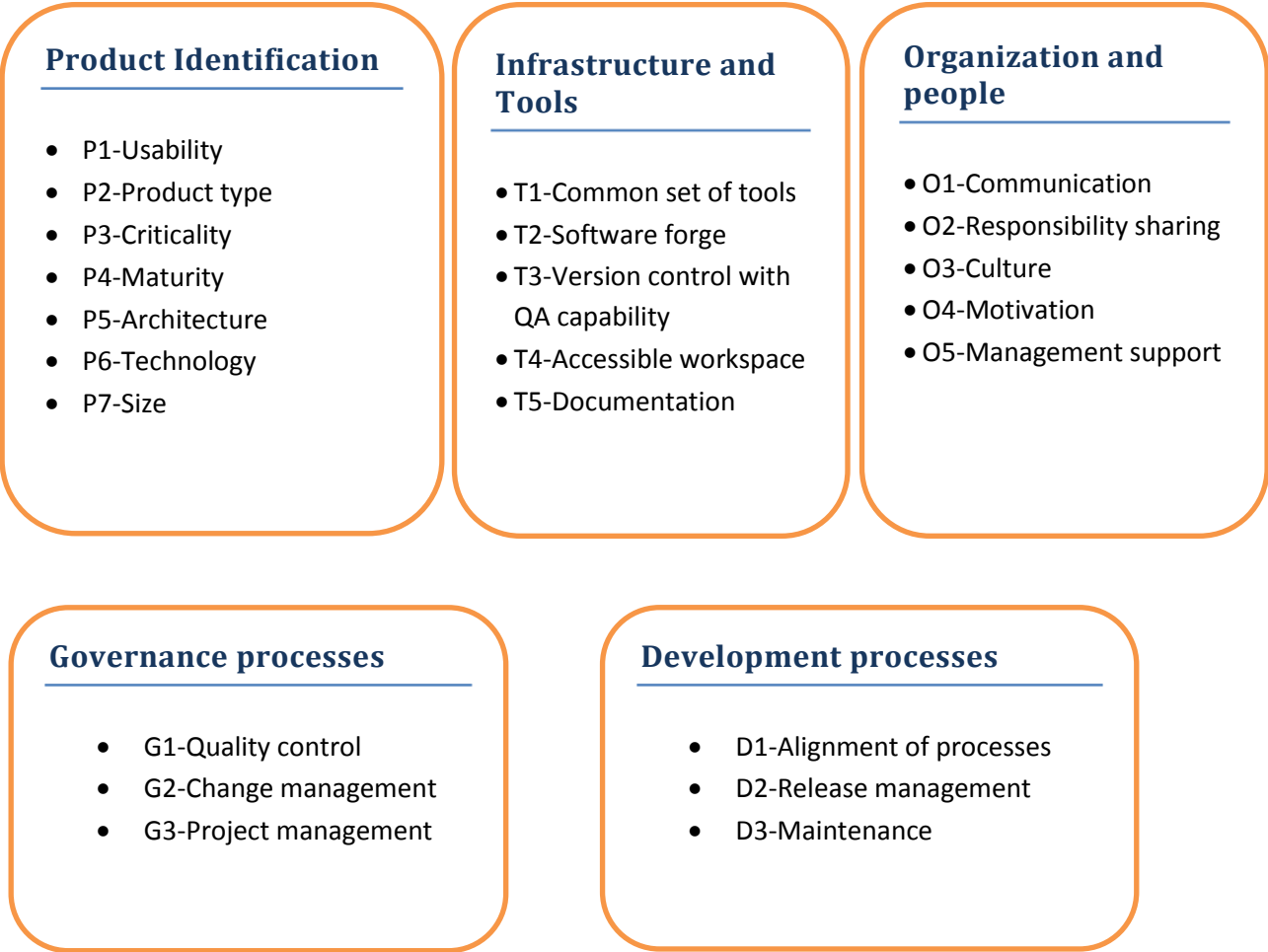


Figure 10- Adoption framework by the company

Most of the available frameworks or guidelines for ISS focus only on some aspects of Inner Source for example the framework proposed by Sharma et al. is mainly concerned about the community in Inner Source [55]. The areas of focus include community building, community governance and community infrastructure. There is no discussion on product characteristics. Community infrastructure guidelines F8 to F11 Table 4 are in accordance with the results of our study. In addition, electronic communication(F4) and having a reward system (F7) was also recommended by participants in this study.

Torkar et al. study focuses on developers and the community as well and how to facilitate the developers' collaboration in FLOSS project. The purpose is to introduce guidelines and adoption opportunities rather than a comprehensive framework [56]. We found some common parts between our studies:

Torkar et al. study [56]	Correspondent points in our study
Reduce technical debt	G1: Quality control activities and tools
Define an entry path for newcomers	O2: Mentor role-T5: Documentation
Increase information availability and visibility	T2: Forge-Dashboard
Embrace asynchronous tools for communication and decision making	O1: Communication and information sharing
Let developers influence ways of working	D1: Process alignment, O2: responsibility sharing based on merit
Allow task selection	O2: Processes: Self task selection

Table 8- Comparison between our study and Torkar et al. [56]

Stol et. al is perhaps the closest study to ours as it considers different aspects of software development and provides guidelines in different areas [50].

Below we provide a comparison of the findings in our studies.

Stol et al study [18]			Our study
Software Product	SP1- Runnable software product	There should be an Initial set of components or product that is runnable and usable	P5-Maturity
	SP2- Needed by several stake holders	Needed by several stakeholders	P1- Usability
	SP3- Maturity State	Requirement and features not fully known at the outset. Need to evolve continuously	P5-Maturity
	SP4- Utility vs. simplicity	Balance between a component functionality and simplicity	-
	SP5- Modularity of software product	Should be highly modular to enable parallel development of the code	P6-Architecture
Development practices	DP1- Requirements elicitation	Requirements are asserted after facts	-
	DP2-Implementation and quality control	Truly independent peer review	D1- Alignment of processes G1-Quality control
	DP3-Release management	Release early release often	D2-Release management
	DP4- Maintenance	Maintenance vs reinvention	D3- Maintenance

Stol et al study [18]			Our study
Tools and Infrastructure	TL1- Development tools	Common tools	T1- Common set of tools
	TL2 -Infrastructure for open access	Infrastructure to facilitate open access to everyone in the organization	T4- Accessible workspace
Organization and community	OC1-work coordination	Task selection vs. assignment	O2-Responsibility sharing
	OC2- Communication	Electronic and asynchronous communication	O1-Communication T5- Documentation
	OC3-Leadership & decision making	Core team reputation and meritocracy	O2-Responsibility sharing
	OC4-Motivation and incentives	Start small, demonstrate value, facilitate change	O4-Motivation
	OC5- Open development culture	Promote an open development culture	O3-Culture
	OC6- Organizational support	Need to have backing from management	O5-Management support

Table 9-comparison between my study and Stol et al. [18] study

As can be seen in the table above a lot of common items are found. However, in our study interviewees did not mention any difference in requirement elicitation in Inner Source. In addition, in Stol et al's product selection criteria, functionality-simplicity balance was a criterion but in our study criteria such as product nature, size and technology were identified. Overall no contradiction with Stol et al. report was found.

5.3.3. Relation between the proposed framework and reported challenges

One of the goals of this study was to propose a framework to address the challenges pointed out by employees. We tried to map the identified challenges to the relevant practices and approaches which were suggested by interviewees. The below table shows the challenges and recommended practices:

Challenges		Recommended Approach
Technical Issues	Tools	Use appointed set of tools from the beginning of project
	Architecture of the existing products	Not suitable for Inner Source. Modular design such as Microservices
	Lack of documentation on many products	Wiki, core team ensures documentation is done in parallel with development
	Access issues	Accessible workspace, use of software forge to centralize work items
	Formation of fork versions	Change management
	Poor quality code	subversion with QA capability, forming QA teams to review code, using automated tests, code review tools and extensions ,Use of bug tracking tools helps keeping track of changes
	Security	Security persistent solutions
Organizational Challenges	Managing and coordinating people	core team, incentive system, responsibility sharing
	Code ownership	Ownership by core team for project based and by individual project in infrastructure based
	Maintenance responsibilities	Devops
	Lack of support from managers	Cultural shift, Support from managers
	Project planning	Alignment of processes, common WBS system for cost control
	Constant reorganizations	Increase motivation, incentive system, transparent and nonhierarchical decision making, self-task selection
People and Individuals' Challenges	Lack of motivation for contribution	Management support promote motivation factors ex reward system
	Communication and coordination issues	core team, use of transparent electronic communication
	Lack of awareness of the existing project	Forge dashboard
	Lack of domain knowledge	Documentation, mentor role

Table 10- Relation between the proposed framework and reported challenges

6. Conclusions

This research aimed to address three research questions about Inner Source. A qualitative research performed as case study in a large company with several development tracks. Three main research questions have been discussed. First one is “What are the obstacles and challenges of adopting Inner Source?”. Semi structured interview was selected as data collection method. The challenges have been summarized and categorized in five groups: Organizational, People, Product, Infrastructure and Process challenges. Identification of the challenges contribute to selection of the practices and processes. The challenges might not be limited to the above ones, but the main ones are stated and could help organizations in taking them into consideration if they choose to opt Inner Source.

The second research question is “Which software components/products are best candidate for inner-sourcing?”. The method selected to identify the criteria was interview which was a good choice. Then, participants voted on the criteria found to reach a consensus. The identified criteria were usability, product nature, criticality, maturity, architecture, technology. The results were then compared to the existing studies to see if they confirm the previous work. Some of the criteria found were mentioned in other studies while few such as technology or product nature were identified in this research.

Finally, to identify the products/components which meet the criteria, workshops were setup with the team members to find the best suitable candidates.

The third research question is “How can the company adopt Inner Source?” In-depth interviews were carried out with representatives and participants from several teams to explore their view on implementing Inner Source. The findings were summarized and presented in section 4.3. Two models for Inner Source adoption was suggested in Section 5.3.1 depending on the product. In Section 5.3.3. strategies and practices to implement Inner Source in an efficient way to overcome the challenges mentioned before were presented. It is very important to emphasize that the adoption model is suited to the needs of the organization and challenges and obstacles mentioned by employees are considered in the suggested adoption model.

As can be seen from the results, not many technical shifts are required for implementing the Inner Source though and most of the changes are aimed at processes, organization and people. None of the techniques or practices are new nor specific to Inner Source rather most are already being used in the software industry. However, some are new to the company. Therefore training and education is needed to establish those practices in the company.

The adoption of Inner Source requires many changes such as cultural and mindset change of employees and managers. Therefore, we suggest starting it in smaller scale and after gaining experience and evaluating the results, use the lessons learnt to extend it to other departments.

6.1. Threats to validity

6.1.1. Construct validity

Too broad definition of constructs is a threat to validity in this study. RQ3 is “how to adopt Inner Source at the company?” It is very broad and some of the processes referred in this thesis needs more elaboration for example development processes or project management approaches were discussed briefly by interviewees, but each of them is a broad area and unfortunately elaborating them separately was not possible in this study due to time constraints.

6.1.2. Internal Validity

Selection bias is a threat to internal validity. We selected totally 16 people from different teams with different roles which were familiar with Open Source. Some of them had Inner Source experience and some were totally new to the concept. Selection of participants in the case study could be a threat to internal validity.

Extraneous variable is another threat to internal validity. We identified list of challenges which lead to lack of contribution. This causal relationship could have been affected by other hidden factors that were unknown to the researcher.

6.1.3. External validity

Setting bias is a threat to external validity. The study was carried out in a large IT company with well-defined processes and tools. The results could vary if the study is carried out in a different context for example a small company which uses different processes with different maturity level. It must be considered that the purpose of case study is not to generalize the results but to study a case in a specific setting.

6.1.4. Reliability

We believe that the data and analysis done is not dependent on the researcher. Processes for data collection and analysis were clearly defined. Results are documented, and recordings are available. Future studies in the same setting should produce consistent results.

6.2. Implications for research

RQ3 covers a broad area and some of the process such as development processes and project management in Inner Source were touched upon in this study, but they require deeper examination and worth a study on their own. The suggested adoption framework leaves a lot of room for investigation for researchers. In addition, criteria for product identification was identified which requires to be validated through more empirical data.

Evaluation of the implementation of Inner Source in the company was out of the scope of this research but it would be of interest to perform a post-implementation study to evaluate the findings.

Whether the results of this study could be generalized to other companies could also be an interesting topic to explore.

6.3. Implications for practitioners

This study has contributed to the existing body of knowledge by providing a comprehensive list of challenges in Inner Source from different aspects and suggesting approaches to overcome the challenges.

We identified product selection criteria and an adoption model for inner sourcing was also proposed. This study could be used as guideline by other companies and practitioners who are considering adopting Inner Source.

7. References

- [1] Senyard, A., & Michlmayr, M. (2004, November). How to have a successful free software project. In *11th Asia-Pacific Software Engineering Conference* (pp. 84-91). IEEE.
- [2] Munassar, N. M. A., & Govardhan, A. (2010). A comparison between five models of software engineering. *International Journal of Computer Science Issues (IJCSI)*, 7(5), 95.
- [3] Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2017). Agile software development methods: Review and analysis. arXiv preprint arXiv:1709.08439.
- [4] Androutsellis-Theotokis, S., Spinellis, D., Kechagia, M., & Gousios, G. (2011). Open source software: A survey from 10,000 feet. *Foundations and Trends® in Technology, Information and Operations Management*, 4(3-4), 187-347
- [5] Laurent, A. M. S. (2004). Understanding open source and free software licensing: guide to navigating licensing issues in existing & new software. " O'Reilly Media, Inc.",4.
- [6] Mockus, A., Fielding, R. T., & Herbsleb, J. D. (2002). Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 11(3), 309-346.
- [7] O'Reilly, T. (1999). Lessons from open-source software development. *Communications of the ACM*, 42(4), 32-37.
- [8] Paulson, J. W., Succi, G., & Eberlein, A. (2004). An empirical study of open-source and closed-source software products. *IEEE Transactions on Software Engineering*, 30(4), 246-256.
- [9] Casson, T., & Ryan, P. S. (2006). Open standards, open source adoption in the public sector, and their relationship to Microsoft's market dominance. *Standards edge: Unifier or divider*, 87.
- [10] Dalle, J. M., & Jullien, N. (2000). Windows vs. Linux: some explorations into the economics of Free Software. *Advances in Complex Systems*, 3(01n04), 399-416.
- [11] Morgan, E. L. (2000). The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary. *Information Technology and Libraries*, 19(2), 105.
- [12] Fitzgerald, B. (2004). A critical look at open source. *Computer*, 37(7), 92-94.
- [13] Wheeler, S., "Open Source Software "<http://sucs.org/sits/articles/opensource/opensource.pdf>", accessed Aug. 2016.
- [14] Reynolds, C. J., & Wyatt, J. C. (2011). Open source, open standards, and health care information systems. *Journal of medical Internet research*, 13(1), e24.
- [15] Houaich, Y. A., & Belaissaoui, M. (2015, February). Measuring the maturity of open source software. In *2015 6th International Conference on Information Systems and Economic Intelligence (SIE)* (pp. 133-140). IEEE.
- [16] Höst, M., Stol, K. J., & Oručević-Alagić, A. (2014). Inner Source project management. In *Software Project Management in a Changing World* (pp. 343-369). Springer, Berlin, Heidelberg, 343-369.
- [17] Stol, K. J., Avgeriou, P., Babar, M. A., Lucas, Y., & Fitzgerald, B. (2014). Key factors for adopting Inner Source. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 23(2), 18.

- [18] Stol, K. J., Babar, M. A., Avgeriou, P., & Fitzgerald, B. (2011). A comparative study of challenges in integrating Open Source Software and Inner Source Software. *Information and Software Technology*, 53(12), 1319-1336.
- [19] Vit Vitharana, P., King, J., & Chapman, H. S. (2010). Impact of internal open source development on reuse: Participatory reuse in action. *Journal of Management Information Systems*, 27(2), 277-304.
- [20] Riehle, D., Ellenberger, J., Menahem, T., Mikhailovski, B., Natchetoi, Y., Naveh, B., & Odenwald, T. (2009). Open collaboration within corporations using software forges. *IEEE software*, 26(2), 52-58.
- [21] Dinkelacker, J., Garg, P. K., Miller, R., & Nelson, D. (2002, May). Progressive open source. In *Proceedings of the 24th International Conference on Software Engineering* (pp. 177-184). ACM.
- [22] Lindman, J., Rossi, M., & Marttiin, P. (2008, September). Applying open source development practices inside a company. In *IFIP International Conference on Open Source Systems* (pp. 381-387). Springer, Boston, MA.
- [23] Lindman, J., Riepula, M., Rossi, M., & Marttiin, P. (2013). Open source technology in intra-organisational software development—private markets or local libraries. In *Managing Open Innovation Technologies* (pp. 107-121). Springer, Berlin, Heidelberg.
- [24] Morgan, L., Feller, J., & Finnegan, P. (2011). Exploring Inner Source as a form of intra-organisational open innovation.
- [25] Wesselius, J. (2008). The bazaar inside the cathedral: Business models for internal markets. *IEEE software*, 25(3), 60-66.
- [26] Claps, G. G., Svensson, R. B., & Aurum, A. (2015). On the journey to continuous deployment: Technical and social challenges along the way. *Information and Software technology*, 57, 21-31.
- [27] Standish Newsroom - Open Source" (Press release). Boston. 2008-04-16. Retrieved 2008-09-08
- [28] Stol, Klaas-Jan, and Muhammad Ali Babar. "Challenges in using open source software in product development: a review of the literature." *Proceedings of the 3rd International Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development*. ACM, 2010.
- [29] Lichtman, Marilyn. *Qualitative Research in Education: A User's Guide: A User's Guide*. Sage, 2012.
- [30] Johnson, B., & Christensen, L. (2008). *Educational research: Quantitative, qualitative, and mixed approaches* (p. 34). Thousand Oaks, CA: Sage Publications.
- [31] Lichtman, M. (2006). *Qualitative Research in Education: A User's Guide*. SAGE Publications (CA).
- [32] Mack, N., Woodsong, C., MacQueen, K. M., Guest, G., & Namey, E. (2005). *Qualitative research methods: a data collectors field guide*.
- [33] Hennink, M., Hutter, I., & Bailey, A. (2010). *Qualitative research methods*. Sage.
- [34] Runeson, P., & Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering*, 14(2), 131-164.
- [35] Schell, C. (1992). The value of the case study as a research strategy. *Manchester Business School*, 2, 1-15.

- [36] Neale, P., Thapa, S., & Boyce, C. (2006). Preparing a case study: A guide for designing and conducting a case study for evaluation input. Massachusetts: Pathfinder international.
- [37] Humble, J., & Farley, D. (2010). Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation (Adobe Reader). Pearson Education.
- [38] Goldman, R., & Gabriel, R. P. (2005). Innovation happens elsewhere: Open source as business strategy. Morgan Kaufmann.
- [39] Feller, J., & Fitzgerald, B. (2002). Understanding open source software development (pp. 143-159). London: Addison-Wesley.
- [39] Theunissen, M., Kourie, D., & Boake, A. (2007, October). Corporate-, Agile-and Open Source Software Development: A Witch's Brew or An Elixir of Life?. In IFIP Central and East European Conference on Software Engineering Techniques (pp. 84-95). Springer, Berlin, Heidelberg.
- [41] Linåker, J., Krantz, M., & Höst, M. (2014, December). On infrastructure for facilitation of Inner Source in small development teams. In International Conference on Product-Focused Software Process Improvement (pp. 149-163). Springer, Cham.
- [42] Gaughan, G., Fitzgerald, B., & Shaikh, M. (2009, August). An examination of the use of open source software processes as a global software development solution for commercial software engineering. In 2009 35th Euromicro Conference on Software Engineering and Advanced Applications (pp. 20-27). IEEE
- [43] Van Der Linden, F. (2009). Applying open source software principles in product lines. Upgrade, 10(2), 32-41.
- [44] Gurbani, V. K., Garvert, A., & Herbsleb, J. D. (2010). Managing a corporate open source software asset. Communications of the ACM, 53(2), 155-159.
- [45] Riehle, D., Ellenberger, J., Menahem, T., Mikhailovski, B., Natchetoi, Y., Naveh, B., & Odenwald, T. (2009). Open collaboration within corporations using software forges. IEEE software, 26(2), 52-58.
- [46] Lindman, J., Rossi, M., & Marttiin, P. (2008, September). Applying open source development practices inside a company. In IFIP International Conference on Open Source Systems (pp. 381-387). Springer, Boston, MA.
- [47] Gerrit Code Review - A Quick Introduction. Available at <https://gerrit-documentation.storage.googleapis.com/Documentation/2.14.3/intro-quick.html/> [last accessed on January 06, 2017].
- [48] Melian, C., Ammirati, C. B., Garg, P., & Sevon, G. (2002). Building Networks of Software Communities in a Large Corporation. Technical Report. Hewlett Packard.
- [49] Gurbani, V. K., Garvert, A., & Herbsleb, J. D. (2006, May). A case study of a corporate open source development model. In Proceedings of the 28th international conference on Software engineering (pp. 472-481). ACM.
- [50] Stol, K. J., & Fitzgerald, B. (2014). Inner Source--adopting open source development practices in organizations: a tutorial. IEEE Software, 32(4), 60-67.
- [51] Thesis cover image, Available at <http://blog.blackducksoftware.com/inner-sourcing-adopting-open-source-development-processes-in-corporate-it/> [Last accessed: 2016-05-03]

- [52] Stol, K. J. (2011). Supporting Product Development with Software from the Bazaar (Doctoral dissertation, UNIVERSITY OF LIMERICK).
- [53] Lakhani, K. R., & Wolf, R. G. (2003). Why hackers do what they do: Understanding motivation and effort in free/open source software projects.
- [54] Riehle, D., Ellenberger, J., Menahem, T., Mikhailovski, B., Natchetoi, Y., Naveh, B., & Odenwald, T. (2009). Bringing Open Source Best Practices into Corporations Using a Software Forge. IEEE Software.
- [55] Sharma, S., Sugumaran, V., & Rajagopalan, B. (2002). A framework for creating hybrid-open source software communities. *Information Systems Journal*, 12(1), 7-25.
- [56] Torkar, R., Minoves, P., & Garrigós, J. (2011). Adopting free/libre/open source software practices, techniques and methods for industrial use. *Journal of the Association for Information Systems*, 12(1), 88.
- [57] Fowler, M., & Lewis, J. (2014). Microservices, Available at : <https://martinfowler.com/articles/microservices.html> [last accessed on January 06, 2019].
- [58] Fowler, M., & Lewis, J. (2014). Microservices. ThoughtWorks, Available at: <https://martinfowler.com/microservices/> [Last accessed on January 06, 2019].
- [59] Bacchelli, A., & Bird, C. (2013, May). Expectations, outcomes, and challenges of modern code review. In *Proceedings of the 2013 international conference on software engineering* (pp. 712-721). IEEE Press.
- [60] Fowler M. Continuous Integration , Available at : <https://www.martinfowler.com/articles/continuousIntegration.html> [last accessed February 2, 2019]
- [61] Agile Manifesto. Available at: <http://agilemanifesto.org/principles.html> / [last accessed on January 06, 2019].

8. Appendixes

8.1. Appendix 1: Successful Open source projects

Source: Open Source Initiative: (<http://royal.pingdom.com/2009/05/29/the-8-most-successful-open-source-products-ever/>)



Open source in itself is a success story. From being a niche concept, it has become a mainstream movement (well, more or less) and has received the attention of both individuals and businesses worldwide. There are thousands of open source projects and products out there, but which ones are the most successful? By successful we mean widely used and widely known. While there are many successful open source products, a few stand head and shoulders above the rest. We have listed them here below.

LINUX



Why it is a success: Linux, hand in hand with GNU software as GNU/Linux, has come a long way since Linus Torvalds announced that he was creating an OS kernel based on Minix back in 1991. These days, a majority of web servers run Linux, and with Ubuntu (see below) it is also (finally) starting to make inroads into the desktop market, and maybe it will soon also be strong player in the mobile market with Android (which uses the Linux kernel).

UBUNTU



Why it is a success: Launched in 2004, Ubuntu is by far the most popular Linux distribution today, especially on the desktop side. Considering the massive success of Ubuntu in recent years, we thought it was worth its own mention here even though we already mentioned Linux.

BSD



Why it is a success: FreeBSD, OpenBSD and NetBSD have been well-respected server OS alternatives for a long time. Derived from Berkeley Unix in the 1990s, we chose to put them into one group here. As an interesting aside, the core for Apple's Mac OS X is derived from FreeBSD.

MYSQL



Why it is a success: MySQL is the most widely used database server in the world, used by a huge amount of websites and services (examples include Wikipedia, Facebook and, more modestly, our very own Pingdom.com...). It's the M in the hugely popular LAMP stack (Linux, Apache, MySQL, PHP).

APACHE



Why it is a success: The Apache HTTP Server has been the most popular web server software in the world since 1996, which is also the year it got started. Apache still has a strong lead, outclassing second runner up IIS in terms of number of deployed websites (according to Netcraft, Apache is currently used by 46% of all websites, while IIS is used by 29%). In 2009 it passed a huge milestone, becoming the first web server to be used by more than 100 million websites.

FIREFOX



Why it is a success: Mozilla's crowning achievement so far, the Firefox web browser has become a mega success. Firefox 1.0 was launched in 2004 and the browser has since then taken away a huge chunk of the browser market from the previously dominant Internet Explorer, and is arguably the reason that Microsoft started to put more effort into updating IE with new versions. Although Firefox is still number two overall, it has become the dominant browser among the more "techie" crowd (this blog, for example, gets 59% of its visits from Firefox and just 18% from IE).

WORDPRESS



Why it is a success: Since its launch in 2004 as a fork of the b2 blog software, WordPress has become a dominant and hugely popular blog platform. In a survey we made back in January, 27% of the top 100 blogs ran on WordPress. If you also counted WordPress.com, Automattic's hosted WordPress service, that number rose to 32%, more than any other blog software. Since then there have also been some changes, such as the nine Wired blogs in the top 100 switching from Typepad to WordPress.com, so that percentage is likely significantly higher now (all else unchanged, it would be 41%).

BIND



Why it is a success: BIND (*The Berkeley Internet Name Domain Server*) is the most widely used DNS server software on the Internet. The first version of BIND goes all the way back to the early 1980s and has been the main DNS server on UNIX systems ever since. It can justly be called the world's de facto standard DNS server.

8.2. Appendix 2: Questionnaire (Guideline for interviews)

Investigate the possibility of adopting Inner Source in the company: A case study

1- Background

Open-source software (OSS) is software with its source code. It is available with a license that allows everyone to download, use and change the software. The source code is not hidden from users and they are considered as beta testers. Today there are many successful examples of OSS available such as Linux, Apache server, Mozilla, etc. It is reported that adopting open source software resulted in saving of 60 billion dollars per year. Besides, adopting OSS has many benefits including low cost, increased quality, involvement of skilled developers from all around the world which means more people are monitoring the project, learning from the community, etc. On the other hand, it involves many challenges such as product documentation, support and maintenance, integration, migration, etc.

Applying Open Source Software development practices and tools within a company is called Inner Source. It is becoming more and more popular nowadays. According to Gaughan et al. the motivations for adopting this approach is the possibility to get support from distributed developers and increase of software reuse and quality as a result of increased software availability and transparency. In addition, everyone in the organization is invited to participate either as developer or contributor which could lead to innovation and increased awareness plus increased speed of development.

However, like open source software development, it is not a straight forward approach. It has many effects on the organization. It affects the way projects are sourced and managed. In addition, the adoption model must match the organization goals and scope. It also affects the operational aspects such as how developers across the company can access and modify the source code. Therefore, companies need to study it carefully before deciding to go for it.

2- Questions

❖ **expected benefits/ Challenges**

1. Has inner-sourcing ever been discussed/tried?
2. What benefits do you expect from Inner-sourcing a product?
3. What are the challenges/obstacles?
4. What could be the motivations for involvement?

❖ **Product**

1. Are there products that are developed mutually/reused by different departments?
2. What product is a good candidate for Inner Source? What characteristics should it have to make it suitable?
 - 2.1. If it is in early stages of development or if it is quite stable?
 - 2.2. What size? Why?
 - 2.3. What Technology? Why?

- 2.4. Usage? Internal/for customer? Why?
- 2.5. Architecture/Modularity? Why?
- 2.6. Is there any product that you can think of as a good candidate?

❖ **Adoption by the company**

1. What development processes do you use today? In which ways Inner Source is different from your current way of working?
2. What infrastructure is required for it? Tools, resources, roles, etc?
3. Which adoption model suits us better? Project based or infrastructure based?
4. What strategies should we have for project management, organization structure, costs sharing, etc.
5. How to manage contributions and coordinating developers?
6. How to ensure quality?
7. Support/maintenance?
8. How do these practices/tools address challenges?

8.3. Appendix 3: Candidate products for inner sourcing

The following products were identified by different teams as a good candidate for Inner-sourcing and they shared most of the characteristics mentioned by interviewees. Information about each product is obtained from the internal documents at the company and their web portal.

- **JVS**

JVS is the framework and reference architecture for Java Enterprise applications at the company. The latest version, JVS4, is the natural evolution of the framework, based on Spring 4 and Java EE 6.

JVS4 adds value in several aspects:

- Makes it possible to build applications using the programming model of Java EE 6 with older Application Servers including concepts like Dependency Injection and Managed Beans.
- Makes it possible to have a development round-trip for deployment in the range of seconds instead of minutes.
- Easy and structured way to handle configuration of different environments like TEST, QA and PROD.

The main theme for JVS4 is to increase both developer- and operational efficiency, taking the total lifecycle cost of JVS software solutions into account.

To support this goal, JVS4 covers the complete end-to-end scope of software development, from business idea to working software in production.

JVS has the following characteristic mentioned by interviewees:

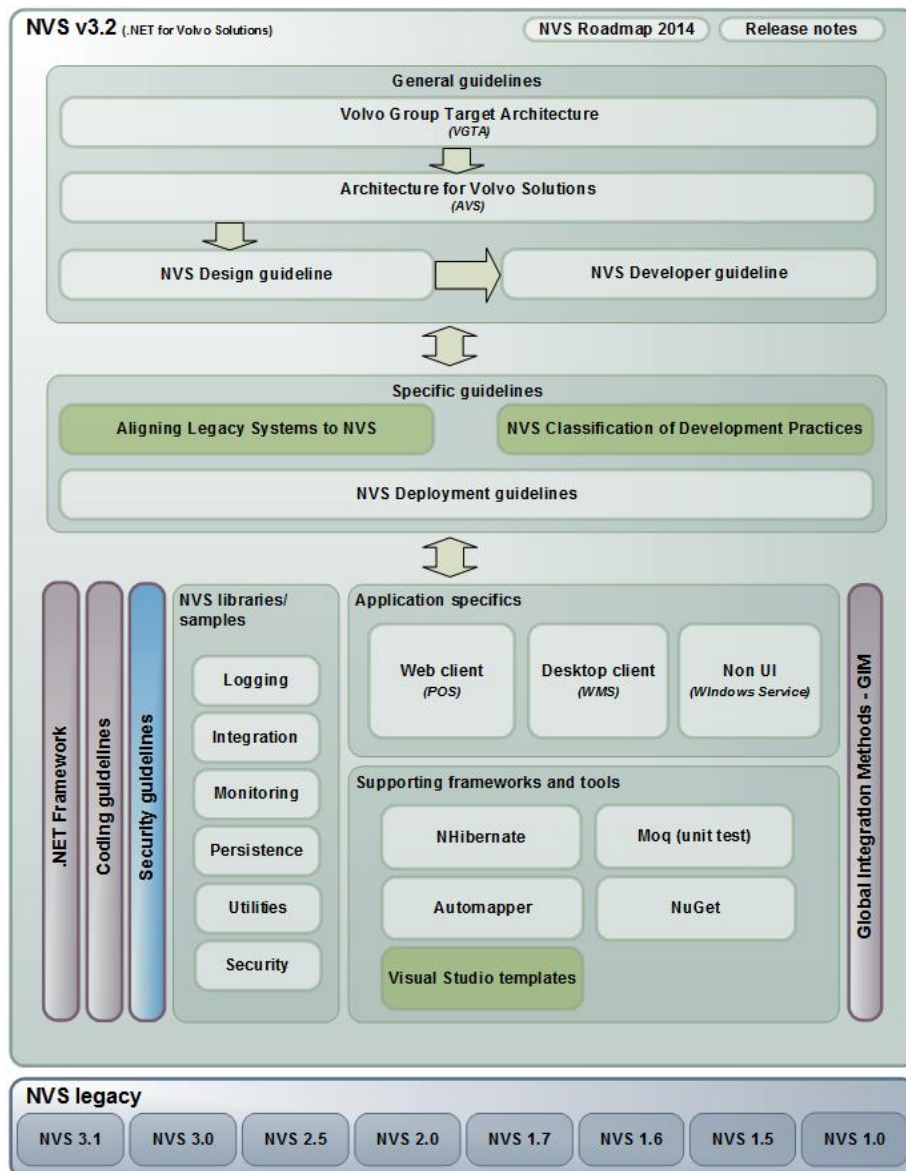
- High usability
- Common Technology Java
- Modular architecture
- Company specific
- Well documented

- **NVS**

NVS is the framework and reference architecture for .Net applications at the company.

As can be seen in the picture below it includes:

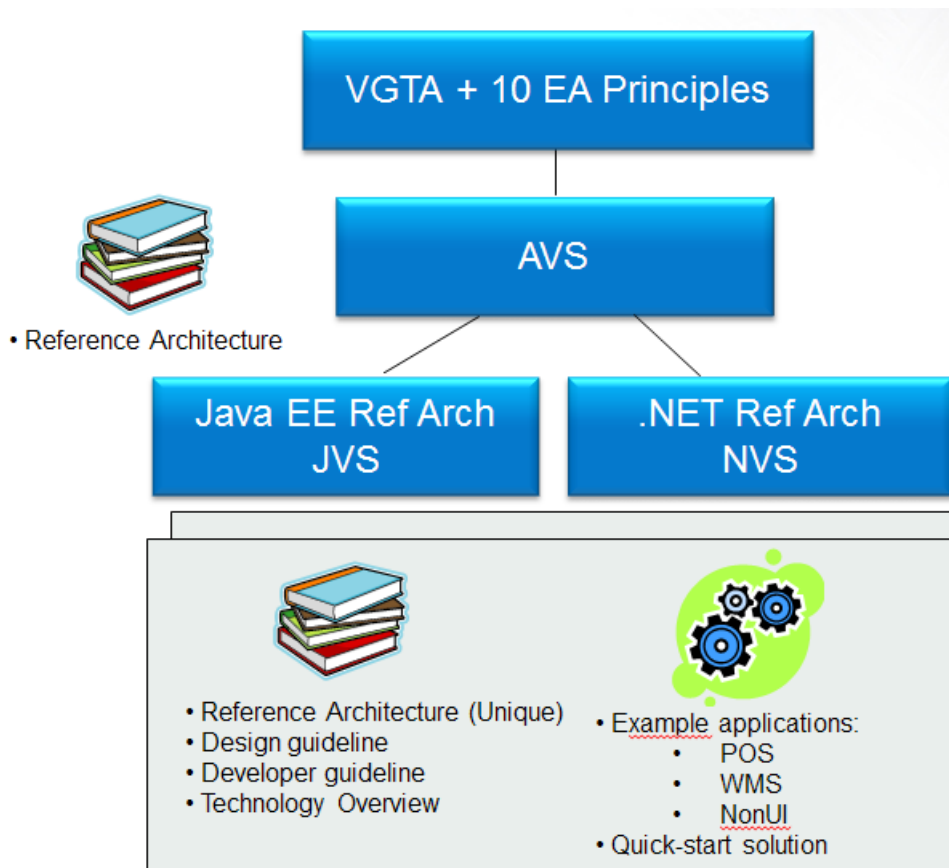
- design and develop guidelines
- Common libraries for security, integration, ...
- Sample applications
- Quick startup solutions



NVS It has the following characteristic mentioned by interviewees:

- High usability
- Common Technology .Net
- Company specific
- Modular architecture
- Well documented

As can be seen in the below picture both JVS and NVS follow the primary reference architecture at the company called AVS and all applications developed at the company must follow its principles.



- **JSVS**

JVS is the Framework and Reference Architecture for JavaScript applications at the company.

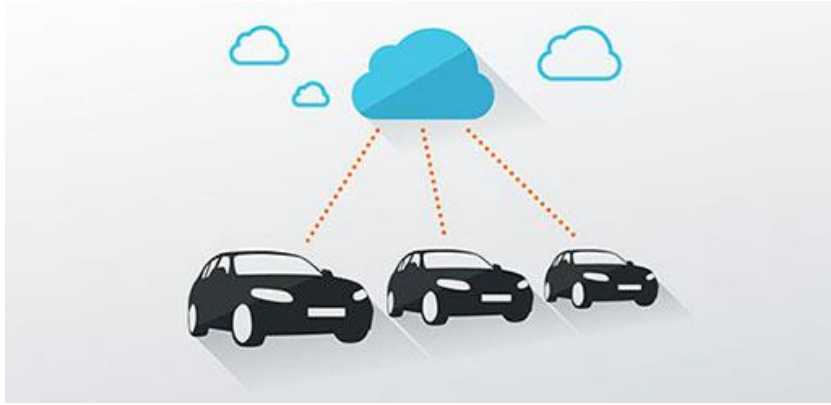
All the above-mentioned frameworks make it easier for quick startup of the development of applications without the need to search for guidelines or tools or libraries. They contain everything that architects and developers need without having to worry about conformity to company standards. In other words, these products serve as a common base for the solutions developed at the company and have a great potential of being Inner Sourced. Even at some point one of the products were tested for Inner-sourcing.

JVS has the following characteristic which interviewees mentioned:

- High usability
- Common Technology JavaScript
- Company specific
- Modular architecture
- Well documented
- Initial phases

- **Common platform by VGT**

Telematics is an area of technology based on vehicles being wirelessly connected. It enables cars and commercial vehicles to do everything from calling for help, connecting fleets, saving fuel, track stolen vehicles to preheating your car. Existing services can be enhanced, and entirely new services can be created using this platform.



It has the following characteristic mentioned by interviewees:

- Company specific
- High usability
- Common Technology Java
- Modular architecture-Micro Services
- Well documented